IBM WebSphere Application Server Network Deployment for IBM i, Version 8.0

Scripting various types of applications

Note

Before using this information, be sure to read the general information under “Notices” on page 707.

Compilation date: July 31, 2011

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with http:// work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Scripting for batch applications

You can use wsadmin scripts to remove the common batch container from your deployment target or redeploy the common batch container on your deployment target. You can use the JobSchedulerCommands command group to show or modify job scheduler attributes, and to create, modify, remove, or list job scheduler custom properties.

Scripting batch applications

After you install the product, you can use scripts to complete various tasks.

removePGC.py

You can use the removePGC.py Jython script to remove the common batch container from your deployment target.

Purpose

The removePGC.py script is provided with the product. The removePGC.py script removes the common batch container from your deployment target or removes it when your deployment target is the only target.

Location

At installation, removePGC.py is copied onto the installation target machines in the <*install_root*>/bin directory.

Usage

To run removePGC.py script with the wsadmin utility, use this command:

```
wsadmin -lang jython -f removePGC.py <option>
```

You might have to modify the wsadmin command to wsadmin.sh or wsadmin.bat, depending on your operating system environment.

To see a list of all available operations, use the following command:

```
wsadmin -lang jython -f removePGC.py --help
```

Operations

- - list
Lists the targets that have the common batch container.

Example

Use following command to list the targets which have the common batch container:

```
wsadmin -lang jython -f removePGC.py --list
```

For example:

```
>> wsadmin -lang jython -f removePGC.py --list
INFO: Grid Execution Environment was found on following targets:
      cell=myCell,cluster=Endpoint1
      cell=myCell,cluster=Endpoint2
      cell=myCell,node=myNode01,server=server1
```

redeployLRS.py

You can use the redeployLRS.py Jython script to redeploy the job scheduler on your deployment target.

Purpose

The redeployLRS.py script is provided with the product. The redeployLRS.py script redeploys the job scheduler on your deployment target.

Location

At installation, redeployLRS.py is copied onto the installation target in the <*install_root*>/bin directory.

Usage

To run redeployLRS.py script with the wsadmin utility, use this command:

```
wsadmin -lang jython -f redeployLRS.py <option>
```

Modify the wsadmin command to wsadmin.sh or wsadmin.bat, depending on your operating system environment.

To see a list of all available operations, use the following command:

```
wsadmin -lang jython -f redeployLRS.py --help
```

Administrator scripting interfaces

This reference information provides examples of how to complete various tasks using the administrative scripting interfaces.

Scripting interfaces

Use the following scripting interfaces to automate tasks that you might normally accomplish with the administrative console.

- "JobSchedulerCommands command group for the AdminTask object"

JobSchedulerCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure the job scheduler with the wsadmin tool. The commands and parameters in the JobSchedulerCommands command group can be used to manage configuration attributes and custom properties.

Use the following commands to manage the job scheduler:

- "showJobSchedulerAttributes"
- "modifyJobSchedulerAttribute" on page 3
- "createJobSchedulerProperty" on page 4
- "modifyJobSchedulerProperty" on page 4
- "removeJobSchedulerProperty" on page 5
- "listJobSchedulerProperties" on page 6

showJobSchedulerAttributes

The showJobSchedulerAttributes command shows all configuration attributes of the job scheduler.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a list of all attributes of the job scheduler.

Batch mode example usage

- Using Jacl

```
$AdminTask showJobSchedulerAttributes
```

- Using Jython

```
AdminTask.showJobSchedulerAttributes( )
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask showJobSchedulerAttributes
```

- Using Jython:

```
AdminTask.showJobSchedulerAttributes( )
```

modifyJobSchedulerAttribute

The modifyJobSchedulerAttribute command modifies a configuration attribute of the job scheduler.

Target object

None

Required parameters

-name
Specifies the name of the attribute to modify. (String)

The following attributes are supported.

1. datasourceJNDIName (default value is jdbc/lrsched)
2. databaseSchemaName (default value is LRSSCHEMA)
3. deploymentTarget (default value is none)
4. endpointJobLogLocation (default value is ${GRID_JOBLOG_ROOT})
5. enableUsageRecording (default value is false)
6. enableUsageRecordingZOS (default value is false)

Optional parameters

-value
Specifies the value of the attribute. (String) If not specified, the default value for the respective attributes is assigned.

Return value

The command returns the job scheduler object ID.

Batch mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerAttribute
{-name datasourceJNDIName -value "jdbc/ds"}
```

- Using Jython:

```
AdminTask.modifyJobSchedulerAttribute('[-name datasourceJNDIName -value jdbc/ds]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerAttribute {-interactive}
```

- Using Jython:

```
AdminTask.modifyJobSchedulerAttribute ('[-interactive]')
```

createJobSchedulerProperty

The createJobSchedulerProperty command creates custom properties for the job scheduler.

Target object

None

Required parameters

-name
Specifies the name of the custom property to create. (String)

-value
Specifies the value of the custom property. (String)

Optional parameters

-description
Specifies the description of the custom property. (String)

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:

```
$AdminTask createJobSchedulerProperty {-name bjsProp1 -value "bjsprop1"}
```

- Using Jython:

```
AdminTask.createJobSchedulerProperty('[-name bjsProp1 -value bjsprop1]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask createJobSchedulerProperty {-interactive}
```

- Using Jython:

```
AdminTask.createJobSchedulerProperty ('[-interactive]')
```

modifyJobSchedulerProperty

The modifyJobSchedulerProperty command modifies custom properties for the job scheduler.

Target object

None

Required parameters

-name
Specifies the name of the custom property to modify. (String)

-value
Specifies the value of the custom property. (String)

Optional parameters

-description
Specifies the description of the custom property. (String)

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerProperty {-name bjsProp1 -value "bjsprop1"}
```

- Using Jython:

```
AdminTask.modifyJobSchedulerProperty('[-name bjsProp1 -value bjsprop1]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerProperty {-interactive}
```

- Using Jython:

```
AdminTask.modifyJobSchedulerProperty ('[-interactive]')
```

removeJobSchedulerProperty

The removeJobSchedulerProperty command removes custom properties of the job scheduler.

Target object

None

Required parameters

-name
Specifies the name of the custom property to remove. (String)

Optional parameters

None

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:

```
$AdminTask removeJobSchedulerProperty {-name bjsProp1}
```

- Using Jython:

```
AdminTask.removeJobSchedulerProperty('[-name bjsProp1]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask removeJobSchedulerProperty {-interactive}
```

- Using Jython:

```
AdminTask.removeJobSchedulerProperty ('[-interactive]')
```

listJobSchedulerProperties

The listJobSchedulerProperties command lists all of the custom properties the job scheduler.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a list of all of the custom properties of the job scheduler.

Batch mode example usage

- Using Jacl:

```
$AdminTask listJobSchedulerProperties
```

- Using Jython:

```
AdminTask.listJobSchedulerProperties( )
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask listJobSchedulerProperties
```

- Using Jython:

```
AdminTask.listJobSchedulerProperties( )
```

Chapter 2. Scripting for Data access resources

This page provides a starting point for finding information about data access. Various enterprise information systems (EIS) use different methods for storing data. These backend data stores might be relational databases, procedural transaction programs, or object-oriented databases.

The flexible IBM® WebSphere® Application Server provides several options for accessing an information system backend data store:

- Programming directly to the database through the JDBC 4.0 API, JDBC 3.0 API, or JDBC 2.0 optional package API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA-compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

Service Data Objects (SDO) simplify the programmer experience with a universal abstraction for messages and data, whether the programmer thinks of data in terms of XML documents or Java objects. For programmers, SDOs eliminate the complexity of the underlying data access technology such as, JDBC, RMI/IIOP, JAX-RPC, and JMS, and message transport technology such as, java.io.Serializable, DOM Objects, SOAP, and JMS.

Configuring data access with wsadmin scripting

Use these topics to learn about using wsadmin scripting to configure data access.

About this task

This topic contains the following tasks:

Procedure

Configuring a JDBC provider using wsadmin

You can configure a JDBC provider using the wsadmin scripting tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client for more information. .

Procedure

1. There are two ways to perform this task. Perform one of the following:
   - Using the AdminTask object:
     - Using Jacl:

       ```
       $AdminTask createJDBCProvider {-interactive}
       ```

     - Using Jython:

       ```
       AdminTask.createJDBCProvider (['-interactive'])
       ```

   - Using the AdminConfig object:
     a. Identify the parent ID and assign it to the node variable. The following example uses the node configuration object as the parent. You can modify this example to use the cell, cluster, server, or application configuration object as the parent.
        - Using Jacl:

          ```
          set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
          ```

        - Using Jython:

          ```
          node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
          print node
          ```

          Example output:

          ```
          mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
          ```

     b. Identify the required attributes:

        Fast path: For supported JDBC drivers, you can also script JDBC providers according to the same pre-configured templates that are used by the administrative console logic. Consult the article Creating configuration objects using the wsadmin tool for details.

        - Using Jacl:

          ```
          $AdminConfig required JDBCProvider
          ```

        - Using Jython:

          ```
          print AdminConfig.required('JDBCProvider')
          ```

          Example output:

          ```
          Attribute     Type
          name     String
          implementationClassName   String
          ```

     c. Set up the required attributes and assign it to the jdbcAttrs variable. You can modify the following example to setup non-required attributes for JDBC provider.
        - Using Jacl:

          ```
          set n1 [list name JDBC1]
          set implCN [list implementationClassName myclass]
          set  jdbcAttrs [list  $n1  $implCN]
          ```

          Example output:

          ```
          {name {JDBC1}} {implementationClassName {myclass}}
          ```

        - Using Jython:

```
n1 = ['name', 'JDBC1']
implCN = ['implementationClassName', 'myclass']
jdbcAttrs = [n1,  implCN]
print jdbcAttrs
```

Example output:

```
[['name', 'JDBC1'], ['implementationClassName', 'myclass']]
```

d. Create a new JDBC provider using node as the parent:

– Using Jacl:

```
$AdminConfig create JDBCProvider $node $jdbcAttrs
```

– Using Jython:

```
AdminConfig.create('JDBCProvider', node, jdbcAttrs)
```

Example output:

```
JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
```

2. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
3. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

What to do next

If you modify the class path or native library path of a JDBC provider: After saving your changes (and synchronizing the node in a network deployment environment), you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Configuring new data sources using wsadmin

You can configure new data sources using the wsadmin scripting tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client for more information.

In WebSphere Application Server, any JDBC driver properties that are required by your database vendor must be set as data source properties. Consult the article Data source minimum required settings, by vendorVendor-specific data sources minimum required settings in the *Troubleshooting and support* PDF to see a list of these properties and setting options, ordered by JDBC provider type. Consult your database vendor documentation to learn about available optional data source properties. Script them as *custom properties* after you create the data source. In the Related links section of this article, click the "Configuring new data source custom properties using scripting" link for more information.

About this task

There are two ways to perform this task; use either of the following wsadmin scripting objects:

- AdminTask object
- AdminConfig object

AdminConfig gives you more configuration control than the AdminTask object. When you create a data source using AdminTask, you supply universally required properties only, such as a JNDI name for the data source. (Consult the article JDBCProviderManagement command group for the AdminTask object for more information.) Other properties that are required by your JDBC driver are assigned default values by Application Server. You cannot use AdminTask commands to set or edit these properties; you must use AdminConfig commands.

Procedure

- Using the AdminConfig object to configure a new data source:
  1. Identify the parent ID, which is the name and location of the JDBC provider that supports your data source.
     - Using Jacl:

       ```
       set newjdbc [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/]
       ```

     - Using Jython:

       ```
       newjdbc = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/')
       print newjdbc
       ```

     Example output:

     ```
     JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
     ```

  2. Obtain the required attributes.

     Fast path: For supported JDBC drivers, you can also script data sources according to the same pre-configured templates that are used by the administrative console logic. For more information, see the topic Creating configuration objects using the wsadmin scripting tool.

     - Using Jacl:

       ```
       $AdminConfig required DataSource
       ```

     - Using Jython:

       ```
       print AdminConfig.required('DataSource')
       ```

     Example output:

     ```
     Attribute   Type
     name    String
     ```

     Tip: If the database vendor-required properties (which are referenced in the topic Data source minimum required settings, by vendor) are not displayed in the resulting list of required attributes, script these properties as data source custom properties after you create the data source.

  3. Set up the required attributes.
     - Using Jacl:

       ```
       set name [list name DS1]
       set dsAttrs [list $name]
       ```

     - Using Jython:

       ```
       name = ['name', 'DS1']
       dsAttrs = [name]
       ```

  4. Create the data source.
     - Using Jacl:

       ```
       set newds [$AdminConfig create DataSource $newjdbc $dsAttrs]
       ```

     - Using Jython:

       ```
       newds = AdminConfig.create('DataSource', newjdbc, dsAttrs)
       print newds
       ```

     Example output:

     ```
     DS1(cells/mycell/nodes/mynode|resources.xml#DataSource_1)
     ```

- Using the AdminTask object to configure a new data source:
  - Using Jacl:

    ```
    $AdminTask createDatasource {-interactive}
    ```

  - Using Jython:

    ```
    AdminTask.createDatasource (['-interactive'])
    ```

- Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
- In a network deployment environment only, synchronize the node. For more information, see the topic Synchronizing nodes with the wsadmin tool.

What to do next

To set additional properties that are supported by your JDBC driver, script them as data source custom properties.

Configuring new connection pools using wsadmin

You can use wsadmin scripting tool to configure new connection pools.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```
   - Using Jython:

     ```
     newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
     ```

   Example output:

   ```
   DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
   ```
2. Creating connection pool:
   - Using Jacl:

     ```
     $AdminConfig create ConnectionPool $newds {}
     ```
   - Using Jython:

     ```
     print AdminConfig.create('ConnectionPool', newds, [])
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#ConnectionPool_1)
   ```
3. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
4. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Changing connection pool settings with the wsadmin tool

You can use the wsadmin scripting tool to change connection pool settings.

About this task

The wsadmin tool runs scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation and configuration, application deployment, and server runtime operations. The product supports the Jacl and Jython scripting languages. To learn more about the wsadmin tool see the topic Starting the wsadmin scripting client.

To use the wsadmin tool to change connection pool settings:

Procedure

1. Launch a scripting command. There are several options for you to run scripting commands, ranging from running them interactively to running them in a profile.

   To change connection pool settings, you use the getAttribute and setAttribute commands, run against the various settings objects.

   For example, to change the connection timeout setting, the commands are:

   ```
   $AdminControl getAttribute $objectname connectionTimeout
   $AdminControl setAttribute $objectname connectionTimeout 200
   ```

   where:
   - $ is a Jacl operator for substituting a variable name with its value
   - getAttribute and setAttribute are the commands
   - `connectionTimeout` is the object whose value you are resetting
2. For Jacl code examples of each of the connection pool settings, see the topic Example: Changing connection pool settings with the wsadmin tool.

Example

Example: Changing connection pool settings with the wsadmin tool. Using the wsadmin AdminControl object, you can script changes to connection pool settings.

The wsadmin tool runs scripts in the Jacl and Jython languages only. You must start the wsadmin scripting client to perform any scripting task. For more information, see the topic Starting the wsadmin scripting client.

For more information about the AdminControl scripting object, see the topic Using the AdminControl object for scripted administration.

Connection Timeout

The Connection timeout can be changed at any time while the pool is active. All connection requests that are waiting for a connection are changed to the new value minus the time already waited, and are returned to the wait state if there are no available connections.

For example, if the connection timeout is changed to 300 seconds and a connection request has waited 100 seconds, the connection request waits for 200 more seconds if no connection becomes available.

```
$AdminControl getAttribute $objectname connectionTimeout
$AdminControl setAttribute $objectname connectionTimeout 200
```

For more information about this setting, see the topic Connection pool settings. The Connection pool settings topic is in the *Administering applications and their environment* PDF.

Maximum Connections

The maximum connections can be changed at any time except in the case where stuck connection support is active.

If stuck connection support is active, an attempt to change the maximum connections is made. If the attempt fails, an `IllegalState` exception occurs. See the topic Connection pool advanced settings for more information.

If stuck connection support is not active, the maximum connections are changed to the new value. If the new value is greater than the current value, the number of connections increases to the new value and any requests waiting are notified. If the new value is less than the current value and Aged Timeout or

Reap Time is used, the number of connections decreases to the new value depending on pool activity. If agedTimeout or Reap Time are not used, no automatic attempt made to reduce the total number of connections. To manually reduce the number of connections to the new maximum connections, use the Mbean function purgePoolContents.

```
$AdminControl getAttribute $objectname maxConnections
$AdminControl setAttribute $objectname maxConnections 200
```

For more information about this setting, see the topic Connection pool settings.

Minimum Connections

The minimum connections can be changed at any time.

```
$AdminControl getAttribute $objectname minConnections
$AdminControl setAttribute $objectname minConnections 200
```

For more information about this setting, see the topic Connection pool settings.

Reap Time

The reap time can be changed at any time. The reap time interval is changed to the new value at the next interval.

```
$AdminControl getAttribute $objectname reapTime
$AdminControl setAttribute $objectname reapTime 30
```

Unused Timeout

The unused timeout can be changed at any time.

```
$AdminControl getAttribute $objectname unusedTimeout
$AdminControl setAttribute $objectname unusedTimeout 900
```

For more information about this setting, see the topic Connection pool settings.

Aged Timeout

The Aged Timeout can be changed at any time.

```
$AdminControl getAttribute $objectname agedTimeout
$AdminControl setAttribute $objectname agedTimeout 900
```

For more information about this setting, see the topic Connection pool settings.

Purge Policy

The purge policy can be changed at any time.

```
$AdminControl getAttribute $objectname purgePolicy
$AdminControl setAttribute $objectname purgePolicy "Failing Connection Only"
```

For more information about this setting, see the topic Connection pool settings.

Surge Protection Support

Surge connection support starts if surgeThreshold is > -1 and surgeCreationInterval is > 0. The surge protection properties can be changed at any time.

```
$AdminControl getAttribute $objectname surgeCreationInterval
$AdminControl setAttribute $objectname surgeCreationInterval 30
$AdminControl getAttribute $objectname surgeThreshold
$AdminControl setAttribute $objectname surgeThreshold 15
```

For more information about this setting, see the topic Connection pool settings.

Stuck connection Support

An attempt is made to change the stuckTime, stuckTimerTime or stuckThreshold properties. If the attempt fails, an `IllegalState` exception occurs. The pool cannot have any active requests or active connections during this request. For the stuck connection support to start, all three stuck property values must be greater than 0, and the maximum connections value must be > 0.

If the connection pool is stuck, you cannot change the stuck or the maximum connection properties. If you are stuck, there are active connections.

```
$AdminControl getAttribute $objectname stuckTime
$AdminControl setAttribute $objectname stuckTime 30
$AdminControl getAttribute $objectname stuckTimerTime
$AdminControl setAttribute $objectname stuckTimerTime 15
$AdminControl getAttribute $objectname stuckThreshold
$AdminControl setAttribute $objectname stuckThreshold 10
```

For more information about this setting, see the topic Connection pool advanced settings.

Test Connection Support (This is only supported for a DataSource)

The test connection support starts when the testConnection property is set to true, and the interval is > 0. The test connection properties can be changed at any time.

```
$AdminControl getAttribute $objectname testConnection
$AdminControl setAttribute $objectname testConnection 30
$AdminControl getAttribute $objectname testConnectionInterval
$AdminControl setAttribute $objectname testConnectionInterval 15
```

For more information about this setting, see the topic Test connection service.

Connection Pool Partition Support

```
$AdminControl invoke $objectname freePoolDistributionTableSize
$AdminControl invoke $objectname numberOfFreePoolPartitions
$AdminControl invoke $objectname numberOfSharedPoolPartitions
$AdminControl invoke $objectname gatherPoolStatisticalData
$AdminControl invoke $objectname enablePoolStatisticalData
```

For more information about this setting, see the topic Connection pool settings.

Show Pool Information Support

The show pool operations can be changed at any time.

```
$AdminControl invoke $objectname showAllPoolContents
$AdminControl invoke $objectname showPoolContents
$AdminControl invoke $objectname showAllocationHandleList
```

Purge Connection Support

PurgePool can be changed at any time.

```
$AdminControl invoke $objectname purgePoolContents normal
$AdminControl invoke $objectname purgePoolContents immediate
```

For both data sources and connection factories, if the normal option is selected, the purged pool behaves as follows after the purge call:

- Existing in-flight transactions continue to work.
- Shared connection requests are honored.
- Free connections are cleaned up and destroyed.

- In-use connections (connections in transactions) are cleaned up and destroyed when returned to the connection pool.
- close() calls issued on any connections obtained prior to the purgePoolContents call are done synchronously (they wait for the jdbc driver to return before proceeding).
- Requests for new connections (not handles to existing old connections) are honored.

For data sources based on the built-in WebSphere relational resource adapter, if the immediate option is selected, the purged pool behaves as follows after the purge call:

- No new transactions are allowed to start on any connections obtained prior to the purgePoolContents call. Instead, a StaleConnectionException is thrown.
- No new handles are allowed to be handed out on any connections obtained prior to the purgePoolContents call. Instead, a StaleConnectionException is thrown.
- Existing in-flight transactions continue to work, but any new activities on the purged connection cause a StaleConnectionException or an XAER_FAIL exception.
- close() calls issued on any connections obtained prior to the purgePoolContents() call are done asynchronously (no wait time).
- Requests for new connections (not handles to existing old connections) are honored.
- The number of connections are decremented immediately. This might cause the total number of connections in the application server to be temporarily out of sync with the total number of connections in the database.

For data sources not based on the built-in WebSphere relational resource adapter and all connection factories, if the immediate option is selected, the purged pool behaves as follows after the purge call:

- close() calls issued on any connections obtained prior to the purgePoolContents() call are done asynchronously (no wait time)
- Requests for new connections (not handles to existing old connections) are honored.
- The number of connections are decremented immediately. This might cause the total number of connections in the application server to be temporarily out of sync with the total number of connections in the backend resource.

Pool Control Support

Pause and resume can be changed at any time.

```
$AdminControl invoke $objectname pause
$AdminControl invoke $objectname resume
```

Example: Accessing MBean connection factory and data sources using wsadmin

Wsadmin commands are used to access connection factory and data source MBeans. MBeans can retrieve or update properties for a connection factory or data source.

To get a list of J2C connection factory or data source Mbean object names, from the wsadmin command-line use one of the following administrative control commands:

```
$AdminControl queryNames *:type=J2CConnectionFactory,*
$AdminControl queryNames *:type=DataSource,*
```

Example output from DataSource query:

```
wsadmin>$AdminControl queryNames *:type=DataSource,*
"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Derby JDBC Provider,j2eeType=JDBCDataSource,J2EESe
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Derby JDBC Provider,cell=system1Node01Cell"
```

By using the J2C connection factory or data source MBean object name, you can access the MBean. The name follows the `webSphere:name=` expression. In the following example, for the first set, the default data source name is set on variable name. For the second set, the object name is set on variable `objectName`.

Example using the Default DataSource:

```
- wsadmin>set name [list Default Datasource]

Default Datasource

- wsadmin>set objectName [$AdminControl queryNames *:name=$name,*]


"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Derby JDBC Provider,j2eeType=JDBCDataSource,J2EESe
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Derby JDBC Provider,cell=system1Node01Cell"
```

Now you can use the objectName for getting help on attributes and operations available for this Mbean.

```
$Help attributes $objectName
$Help operations $objectName
```

To get or set an attribute or to use an operation, use one of the following commands:

```
$AdminControl getAttribute $objectName "attribute name"
$AdminControl setAttribute $objectName "attribute name" value
$AdminControl invoke $objectName "operation"
```

Attribute, operation examples:

```
Get and set an attribute maxConnections. If you get no value, a null value, or
a java.lang.IllegalStateException, the connection factory or data source for this
MBean has not been created. The connection factory or data source is created at first
JNDI lookup. For this example, the Default DataSource must be used before get,
set and invoke will work.

wsadmin>$AdminControl getAttribute $objectName maxConnections
10

wsadmin>$AdminControl setAttribute $objectName maxConnections 20

wsadmin>$AdminControl getAttribute $objectName maxConnections
20

Using invoke

wsadmin>$AdminControl invoke $objectName showPoolContents

PoolManager name:DefaultDatasource
PoolManager object:746354514
Total number of connections: 3 (max/min 20/1, reap/unused/aged 180/1800/0,
connectiontimeout/purge 1800/EntirePool)
                            (testConnection/inteval false/0, stuck timer/time
/threshold 0/0/0, surge time/connections 0/-1)
Shared Connection information (shared partitions 200)
  No shared connections

Free Connection information (free distribution table/partitions 5/1)
  (2)(0)MCWrapper id 5c6af75b  Managed connection WSRdbManagedConnectionImpl@4b5
a775b  State:STATE_ACTIVE_FREE
  (2)(0)MCWrapper id 3394375a  Managed connection WSRdbManagedConnectionImpl@328
5f75a  State:STATE_ACTIVE_FREE
  (2)(0)MCWrapper id 4795b75a  Managed connection WSRdbManagedConnectionImpl@46a
4b75a  State:STATE_ACTIVE_FREE

  Total number of connections in free pool: 3
UnShared Connection information
  No unshared connections
```

Configuring new data source custom properties using wsadmin

You can configure new data source custom properties using the wsadmin tool scripting tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new data source custom property:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```

   - Using Jython:

     ```
     newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
     print newds
     ```

   Example output:

   ```
   DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newds propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newds, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_8)
   ```

3. Get required attribute:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up attributes:
   - Using Jacl:

     ```
     set name [list name RP4]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP4']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

Example output:

```
RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new Java 2 Connector authentication data entries using wsadmin

You can configure new Java 2 Connector (J2C) authentication data entries with the wsadmin scripting tool.

Before you begin

Before starting this task, the wsadmin tool must be running. Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new J2C authentication data entry:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set security [$AdminConfig getid /Cell:mycell/Security:/]
     ```

   - Using Jython:

     ```
     security = AdminConfig.getid('/Cell:mycell/Security:/')
     print security
     ```

   Example output:

   ```
   (cells/mycell|security.xml#Security_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required JAASAuthData
     ```

   - Using Jython:

     ```
     print AdminConfig.required('JAASAuthData')
     ```

   Example output:

   ```
   Attribute       Type
   alias           String
   userId          String
   password        String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set alias [list alias myAlias]
     set userid [list userId myid]
     set password [list password secret]
     set jaasAttrs [list $alias $userid $password]
     ```

     Example output:

     ```
     {alias myAlias} {userId myid} {password secret}
     ```

   - Using Jython:

```
alias = ['alias', 'myAlias']
userid = ['userId', 'myid']
password = ['password', 'secret']
jaasAttrs = [alias, userid, password]
print jaasAttrs
```

Example output:

```
[['alias', 'myAlias'], ['userId', 'myid'], ['password', 'secret']]
```

4. Create JAAS auth data:
   - Using Jacl:

     ```
     $AdminConfig create JAASAuthData $security $jaasAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('JAASAuthData', security, jaasAttrs)
     ```

   Example output:

   ```
   (cells/mycell|security.xml#JAASAuthData_2)
   ```

5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new WAS40 data sources using wsadmin scripting

Use scripting to configure a new WAS40 data source.

Before you begin

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client using wsadmin scripting” on page 549 article for more information.

About this task

Perform the following steps:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjdbc [$AdminConfig getid "/JDBCProvider:Apache Derby JDBC Provider/"]
     ```

   - Using Jython:

     ```
     newjdbc = AdminConfig.getid('/JDBCProvider:Apache Derby JDBC Provider/')
     print newjdbc
     ```

   Example output:

   ```
   JDBC1(cells/mycell/nodes/mynode|resources.xml$JDBCProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WAS40DataSource
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WAS40DataSource')
     ```

   Example output:

   ```
   Attribute    Type
   name    String
   ```

3. Set up required attributes:
   - Using Jacl:

```
set name [list name was4DS1]
set ds4Attrs [list $name]
```

- Using Jython:

```
name = ['name', 'was4DS1']
ds4Attrs = [name]
```

4. Create WAS40DataSource:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig create WAS40DataSource $newjdbc $ds4Attrs]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.create('WAS40DataSource', newjdbc, ds4Attrs)
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynode|resources.xml#WAS40DataSource_1)
   ```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new WAS40 connection pools using wsadmin scripting

You can use scripting to configure a new WAS40 connection pool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic on starting the wsadmin tool for more information.

About this task

Perform the following steps to configure a new WAS40 connection pool:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynodes:resources.xml$WAS40DataSource_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WAS40ConnectionPool
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WAS40ConnectionPool')
     ```

   Example output:

   ```
   Attribute       Type
   minimumPoolSize     Integer
   maximumPoolSize     Integer
   connectionTimeout   Integer
   idleTimeout        Integer
   orphanTimeout       Integer
   statementCacheSize  Integer
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set mps [list minimumPoolSize 5]
     set minps [list minimumPoolSize 5]
     set maxps [list maximumPoolSize 30]
     set conn [list connectionTimeout 10]
     set idle [list idleTimeout 5]
     set orphan [list orphanTimeout 5]
     set scs [list statementCacheSize 5]
     set 40cpAttrs [list $minps $maxps $conn $idle $orphan $scs]
     ```

     Example output:

     ```
     {minimumPoolSize 5} {maximumPoolSize 30}
     {connectionTimeout 10} {idleTimeout 5}
     {orphanTimeout 5} {statementCacheSize 5}
     ```

   - Using Jython:

     ```
     minps = ['minimumPoolSize', 5]
     maxps = ['maximumPoolSize', 30]
     conn = ['connectionTimeout', 10]
     idle = ['idleTimeout', 5]
     orphan = ['orphanTimeout', 5]
     scs = ['statementCacheSize', 5]
     cpAttrs = [minps, maxps, conn, idle, orphan, scs]
     print cpAttrs
     ```

     Example output:

     ```
     [[minimumPoolSize, 5], [maximumPoolSize, 30],
     [connectionTimeout, 10], [idleTimeout, 5],
     [orphanTimeout, 5], [statementCacheSize, 5]]
     ```

4. Create was40 connection pool:
   - Using Jacl:

     ```
     $AdminConfig create WAS40ConnectionPool $new40ds $40cpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WAS40ConnectionPool', new40ds, 40cpAttrs)
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode:resources.xml#WAS40ConnectionPool_1)
   ```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring custom properties for a Version 4.0 data source using wsadmin scripting

You can use scripting and the wsadmin tool to configure custom properties for a Version 4.0 data source.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic on starting the wsadmin tool.

About this task

Perform the following steps to configure custom properties for a Version 4.0 data source:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

- Using Jython:

```
new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
print new40ds
```

Example output:

```
was4DS1(cells/mycell/nodes/mynodes|resources.xml$WAS40DataSource_1)
```

2. Get required attributes:
   - Using Jacl:

```
set propSet [$AdminConfig showAttribute $new40ds propertySet]
```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(new40ds, 'propertySet')
print propSet
```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_9)
```

3. Optional: Set up attributes for the server name and port number.

   Attention: This step describes optional attributes that might be required by your J2EE Resource Property. Other attributes might be required.
   - Using Jacl to set up the server name:

```
set name [list name "serverName"]
set value [list value db2was.austin.ibm.com]
set rpAttrs1 [list $name $value]
```

   - Using Jython to set up the server name:

```
name = ['name', 'serverName']
rpAttrs1 = [name]
```

   - Using Jacl to set up the port number:

```
set name [list name "portNumber"]
set value [list value 50000]
set rpAttrs2 [list $name $value]
```

   - Using Jython to set up the port number:

```
name = ['name', 'portNumber']
rpAttrs2 = [name]
```

4. Create J2EE Resource Property:
   - Using Jacl:

```
$AdminConfig create J2EEResourceProperty $propSet $rpAttrs1
$AdminConfig create J2EEResourceProperty $propSet $rpAttrs2
```

   - Using Jython:

```
print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs1)
print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs2)
```

   Example output:

```
serverName(cells/cell_name|resources.xml#J2EEResourceProperty_1236708692906)
serverName(cells/cell_name|resources.xml#J2EEResourceProperty_1236708728281)
```

5. Save the configuration changes.
6. Synchronize the node.

Configuring new J2C resource adapters using wsadmin scripting

Use the wsadmin scripting tool to configure Java 2 Connector resource adapters with Resource Adapter Archive (RAR) files. A RAR file provides the classes and other code to support a resource adapter for access to a specific enterprise information system (EIS), such as the Customer Information Control System (CICS®). Configure resource adapters for an EIS only after you install the appropriate RAR file.

Before you begin

A RAR file, which is often called a Java Connector Architecture (JCA) connector, must comply with the JCA 1.5 and 1.6 Specification. For resource adapters to support JCA Version 1.6, there is added support for Java annotations in RAR modules. For more information on annotation support and metadata, see the topic, JCA 1.6 support for annotations in RAR modules.

Meet these requirements by using a supported assembly tool (as described in the Assembly tools article) to assemble a collection of Java archive (JAR) files, other runnable components, utility classes, and so on, into a deployable RAR file. Then you are ready to install your RAR file in Application Server.

There are two ways to complete this task. This topic uses the AdminConfig object to install resource adapters. Alternatively, you can use the installJ2CResourceAdapter script in the AdminJ2C script library to install a J2C resource adapter in your configuration, as the following example demonstrates:

```
AdminJ2C.installJ2CResourceAdapter("myNode", "C:\temp\jca15cmd.rar", "J2CTest")
```

The scripting library provides a set of procedures to automate the most common administration functions. You can run each script procedure individually, or combine several procedures to quickly develop new scripts.

Procedure

1. Launch the wsadmin script. See the topic Starting the wsadmin scripting client article for more information.
2. Identify the configuration ID of the node to which the resource adapter is installed, as the following examples demonstrate:
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```
   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```
3. Identify the optional attributes.

   The J2CResourceAdapter object does not require specific arguments. Use the following command to display the optional attributes for the J2CResourceAdapter object:
   - Using Jacl:

     ```
     $AdminConfig defaults J2CResourceAdapter
     ```
   - Using Jython:

     ```
     print AdminConfig.defaults('J2CResourceAdapter')
     ```

   The following displays the command output that displays each optional attribute and the data type for the attribute, and denotes the default attributes:

   ```
   Attribute                 Type                        Default
   name                      String
   description               String
   classpath                 String
   nativepath                String
   providerType              String
   isolatedClassLoader       boolean                     false
   archivePath               String
   threadPoolAlias           String                      Default
   singleton                 boolean                     false
   hACapability              ENUM                        RA_NO_HA
   isEnableHASupport         boolean                     false
   propertySet               J2EEResourcePropertySet
   jaasLoginConfiguration    JAASConfigurationEntry
   ```

```
deploymentDescriptor            Connector
connectionDefTemplateProps      ConnectionDefTemplateProps
activationSpecTemplateProps     ActivationSpecTemplateProps
j2cAdminObjects                 J2CAdminObject
adminObjectTemplateProps        AdminObjectTemplateProps
j2cActivationSpec               J2CActivationSpec
properties                      Property
```

4. Set up the attributes of interest.

   Determine the attributes to configure for the J2C resource adapter. In the following examples, the commands set the RAR file path to the `rarFile` variable and the name and description configuration options to the `option` variable:

   - Using Jacl:

     ```
     set rarFile /currentScript/cicseci.rar
     set option {-rar.name  RAR1 -rar.desc "New resource adapter"}
     ```

   - Using Jython:

5. Create a resource adapter.

   Use the installResourceAdapter command for the AdminConfig object to install the resource adapter with the previously set configuration options, as the following examples demonstrate:

   - Using Jacl:

     ```
     $AdminConfig installResourceAdapter $rarFile mynode $option
     ```

   - Using Jython:

     ```
     AdminConfig.installResourceAdapter(rarFile, 'mynode', option)
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring custom properties for J2C resource adapters using wsadmin

You can configure custom properties for Java 2 Connector resource adapters using the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a J2C resource adapters:

Procedure

1. Identify the parent ID and assign it to the newra variable.

   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```

   - Using Jython:

     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newra propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newra, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_8)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP4]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP4']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
   ```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new J2C connection factories using wsadmin scripting

Use the wsadmin scripting tool to configure new Java 2 Connector connection factories.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new J2C connection factory:

Procedure

1. Identify the parent ID and assign it to the newra variable.
   - Using Jacl:

```
set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
```

- Using Jython:

```
newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
print newra
```

Example output:

```
RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
```

2. There are two ways to configure a new J2C connection factory. Perform one of the following:
   - Using the AdminTask object:
     a. List the connection factory interfaces:
        - Using Jacl:

          ```
          $AdminTask listConnectionFactoryInterfaces $newra
          ```

        - Using Jython:

          ```
          AdminTask.listConnectionFactoryInterfaces(newra)
          ```

          Example output:

          ```
          javax.sql.DataSource
          ```

     b. Create a J2CConnectionFactory:
        - Using Jacl:

          ```
          $AdminTask createJ2CConnectionFactory $newra { -name cf1
          -jndiName eis/cf1 -connectionFactoryInterface
          avax.sql.DataSource
          ```

        - Using Jython:

          ```
          AdminTask.createJ2CConnectionFactory(newra, '['-name', 'cf1',
          '-jndiName', 'eis/cf1', '-connectionFactoryInterface',
          'avax.sql.DataSource']')
          ```

   - Using the AdminConfig object:
     a. Identify the required attributes:
        - Using Jacl:

          ```
          $AdminConfig required J2CConnectionFactory
          ```

        - Using Jython:

          ```
          print AdminConfig.required('J2CConnectionFactory')
          ```

          Example output:

          ```
          Attribute Type
          connectionDefinition ConnectionDefinition@
          ```

     b. If your resource adapter is JCA1.5 and you have multiple connection definitions defined, it is required that you specify the ConnectionDefinition attribute. If your resource adapter is JCA1.5 and you have only one connection definition defined, it will be picked up automatically. If your resource adapter is JCA1.0, you do not need to specify the ConnectionDefinition attribute. Perform the following command to list the connection definitions defined by the resource adapter:
        - Using Jacl:

          ```
          $AdminConfig list ConnectionDefinition $newra
          ```

        - Using Jython:

          ```
          print AdminConfig.list('ConnectionDefinition', $newra)
          ```

     c. Set up the required attributes:
        - Using Jacl:

          ```
          set name [list name J2CCF1]
          set jname [list jndiName eis/j2ccf1]
          set j2ccfAttrs [list $name]
          ```

        - Using Jython:

```
name = ['name', 'J2CCF1']
jname = ['jndiName', 'eis/j2ccf1']
j2ccfAttrs = [name,jname]
```

d. If you are specifying the ConnectionDefinition attribute, also set up the following:
   - Using Jacl:
     ```
     set cdattr [list connectionDefinition $cd]
     ```
   - Using Jython:
     ```
     cdattr = ['connectionDefinition', $cd]
     ```

e. Create a J2C connection factory:
   - Using Jacl:
     ```
     $AdminConfig create J2CConnectionFactory $newra $j2ccfAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('J2CConnectionFactory', newra, j2ccfAttrs)
     ```

   Example output:
   ```
   J2CCF1(cells/mycell/nodes/mynode|resources.xml#J2CConnectionFactory_1)
   ```

3. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
4. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new J2C activation specifications using wsadmin scripting

You can configure new Java 2 Connector activation specifications using scripting and the wsadmin scripting tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client topic for more information.

About this task

Perform the following steps to configure a J2C activation specifications:

Procedure

1. Identify the parent ID and assign it to the `newra` variable.
   - Using Jacl:
     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```
   - Using Jython:
     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:
   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:
        ```
        $AdminTask listMessageListenerTypes $newra
        ```
        Using Jython:
        ```
        AdminTask.listMessageListenerTypes(newra)
        ```
        Example output:

```
javax.jms.MessageListener
```

b. Create a J2C administrative object:

Using Jacl:

```
$AdminTask createJ2CActivationSpec $newra { -name ac1
-jndiName eis/ac1 -messageListenerType
javax.jms.MessageListener}
```

Using Jython:

```
AdminTask.createJ2CActivationSpec(newra, ['-name', 'ao1',
'-jndiName', 'eis/ao1', '-messageListenerType',
'javax.jms.MessageListener'])
```

- Using the AdminConfig object:

a. Using Jacl:

```
$AdminConfig required J2CActivationSpec
```

Using Jython:

```
print AdminConfig.required('J2CActivationSpec')
```

Example output:

```
Attribute Type
activationSpec ActivationSpec@
```

b. If your resource adapter is JCA V1.5 and you have multiple activation specifications defined, it is required that you specify the activation specification attribute. If your resource adapter is JCA V1.5 and you have only one activation specification defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the activationSpec attribute. Perform the following command to list the activation specifications defined by the resource adapter:

Using Jacl:

```
$AdminConfig list ActivationSpec $newra
```

Using Jython:

```
print AdminConfig.list('ActivationSpec', $newra)
```

c. Set the administrative object that you need to a variable:

Using Jacl:

```
set ac [$AdminConfig list ActivationSpec $newra]
set name [list name J2CAC1]
set jname [list jndiName eis/J2CAC1]
set j2cacAttrs [list $name $jname $cdcttr]
```

Using Jython:

```
ac = AdminConfig.list('ActivationSpec', $newra)
name = ['name', 'J2CAC1']
jname = ['jndiName', 'eis/j2cac1']
j2cacAttrs = [name, jname,cdattr]
```

d. If you are specifying the ActivationSpec attribute, also set up the following:

Using Jacl:

```
set cdcttr [list activationSpec $ac]
```

Using Jython:

```
cdattr = ['activationSpec', ac]
```

e. Create a J2C activation specification object:

Using Jacl:

```
$AdminConfig create J2CActivationSpec $newra $j2cacAttrs
```

Using Jython:

```
print AdminConfig.create('J2CActivationSpec', newra,j2cacAttrs)
```

Example output:

```
J2CAC1(cells/mycell/nodes/mynode|resources.xml#J2CActivationSpec_1)
```

3. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
4. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new J2C administrative objects using wsadmin scripting

You can use scripting and the wsadmin tool to configure new J2C administrative objects.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic on starting the wsadmin tool.

About this task

Perform the following steps to configure a J2C administrative object:

Procedure

1. Identify the parent ID and assign it to the `newra` variable.
   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```
   - Using Jython:

     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```
2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:

        ```
        $AdminTask listAdminObjectInterfaces $newra
        ```

        Using Jython:

        ```
        AdminTask.listAdminObjectInterfaces(newra)
        ```

        Example output:

        ```
        com.ibm.test.message.FVTMessageProvider
        ```

     b. Create a J2C administrative object:

        Using Jacl:

        ```
        $AdminTask createJ2CAdminObject $newra { -name ao1 -jndiName eis/ao1
        -adminObjectInterface com.ibm.test.message.FVTMessageProvider }
        ```

        Using Jython:

        ```
        AdminTask.createJ2CAdminObject(newra, ['-name', 'ao1', '-jndiName', 'eis/ao1',
        '-adminObjectInterface', 'com.ibm.test.message.FVTMessageProvider'])
        ```
   - Using the AdminConfig object:
     a. Using Jacl:

        ```
        $AdminConfig required J2CAdminObject
        ```

        Using Jython:

        ```
        print AdminConfig.required('J2CAdminObject')
        ```

        Example output:

        ```
        Attribute Type
        adminObject AdminObject@
        ```

b. If your resource adapter is JCA V1.5 and you have multiple administrative objects defined, it is required that you specify the administrative object attribute. If your resource adapter is JCA V1.5 and you have only one administrative object defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the administrative object attribute. Perform the following command to list the administrative objects defined by the resource adapter:

   Using Jacl:

   ```
   $AdminConfig list AdminObject $newra
   ```

   Using Jython:

   ```
   print AdminConfig.list('AdminObject', $newra)
   ```

c. Set the administrative objects that you need to a variable:

   Using Jacl:

   ```
   set ao AdminObjectId
   set name [list name J2CAO1]
   set jname [jndiName eis/j2cao1]
   set j2caoAttrs [list $name $jname]
   ```

   Using Jython:

   ```
   ao = AdminObjectId
   name = ['name', 'J2CAO1']
   set jname = ['jndiName', eis/j2cao1]
   j2caoAttrs = [name, jname]
   ```

d. If you are specifying the AdminObject attribute, also set up the following:

   Using Jacl:

   ```
   set cdattr [list adminObject $ao]
   ```

   Using Jython:

   ```
   cdattr = ['adminObject', ao]
   ```

e. Create a J2C administrative object:

   Using Jacl:

   ```
   $AdminConfig create J2CAdminObject $newra $j2caoAttrs
   ```

   Using Jython:

   ```
   print AdminConfig.create('J2CAdminObject', newra, j2caoAttrs)
   ```

   Example output:

   ```
   J2CAO1(cells/mycell/nodes/mynode|resources.xml#J2CAdminObject_1)
   ```

3. Save the configuration changes.
4. Synchronize the node.

Managing the message endpoint lifecycle using wsadmin scripting

Use the Jython scripting language to manage your message endpoints with the wsadmin tool. Use this topic to query your configuration for message endpoint properties, and to deactivate or reactivate a message endpoint.

About this task

The Java EE Connector Architecture (JCA) allows the application server to link inbound requests from messaging resource adapters to message endpoints. The message endpoint managed bean (MBean) is a Java Management Extensions (JMX) framework MBean that the application server associates with a message endpoint instance.

Use this topic to manage situations where messaging providers fail to deliver messages to their intended destinations. For example, a provider might fail to deliver messages to a message endpoint when its underlying Message Driven Bean attempts to commit transactions against a database server that is not

responding. To troubleshoot the problem, use the wsadmin tool to temporarily disable the message endpoint from handling messages. After troubleshooting the issue, use the wsadmin tool to reactivate the message endpoint.

Also use this topic if you are connecting to WebSphere MQ and you have used the WAS_EndpointInitialState custom property in the activation specification to make a message endpoint start out in a deactivated state. Use the wsadmin tool when you are ready to activate the message endpoint.

The steps in this topic display how to use the AdminControl object and the wsadmin tool to invoke a Message Endpoint MBean to:

- Display properties of the a message endpoint
- Temporarily deactivate a message endpoint
- Reactivate a message endpoint

Note: Starting in Version 7.0, you can use the AdminControl object and the wsadmin tool to deactivate message endpoints to pause the endpoints from receiving messages, and to reactivate message endpoints to resume message handling. If you are connecting to WebSphere MQ, you can also use the WAS_EndpointInitialState custom property in the WebSphere MQ messaging provider activation specification to make a message endpoint start out in a deactivated state. Previously, the application server only activated and deactivated message endpoints when the application or resource adapter was started and stopped.

Procedure

- Display properties of a message endpoint.

  Use the following command to display a list of all message endpoints in your configuration:

```
AdminControl.queryNames('*:type=J2CMessageEndpoint,*')
```

  For the previous example, the command returns the following information for the `JMSMDB_MessageEndpoint`:

```
WebSphere:name=JMSMDB_J2CMessageEndpoint,
process=myServer,
platform=dynamicproxy,
cell=myNode01Cell,
node=myNode01,
version=6.1.0.0,
type=J2CMessageEndpoint,
mbeanIdentifier=cells/myNode01Cell/nodes/myNode01/servers/myServer/resources.xml#example#example.jar#JMSMDB_J2CMessageEndpoint,
spec=1.0,
Server=server1,
diagnosticProvider=false,
j2eeType=JCAMessageEndpoint,
J2EEApplication=example
J2EEServer=myServer,
J2CResourceAdapter=SIB JMS Resource Adapter,
MessageDrivenBean=example#example.jar#JMSMDBActivationSpec=3727AS1
```

- Temporarily deactivate the message endpoint.

  Use the following commands to cache the instance and deactivate the message endpoint:

```
objectName=AdminControl.queryNames('*:name=JMSMDB_MessageEndpoint,*')
AdminControl.invoke(objectName, 'pause')
```

  The command returns the following output:

```
Message Endpoint JMSMDB_J2CMessageEndpoint is deactivated
```

- Reactivate the message endpoint, or activate a message endpoint that started out in a deactivated state.

  Use the following commands to cache the instance and reactivate the message endpoint:

```
objectName=AdminControl.queryNames('*:name=JMSMDB_MessageEndpoint,*')
AdminControl.invoke(objectName, 'resume')
```

  The command returns the following output:

```
Message Endpoint JMSMDB_J2CMessageEndpoint is activated
```

Testing data source connections using wsadmin scripting

You can test connections for data sources with the wsadmin tool and scripting. After you have defined and saved a data source, you can test the data source connection to ensure that the parameters in the data source definition are correct.

About this task

You can use the testConnection command for the AdminControl object to test data source connections for a cell, node, server, application, or cluster scope. This topic provides an example that tests the data source connection for the application scope.

Procedure

- Test the data source connection for a cell, node, or server scope.
  1. Start the wsadmin scripting tool.
  2. Identify the DataSourceCfgHelper MBean and assign it to the dshelper variable.
     - Using Jacl:

       ```
       set ds [$AdminConfig getid /DataSource:DS1/]
       $AdminControl testConnection $ds
       ```

     - Using Jython:

       ```
       ds = AdminConfig.getid('/DataSource:DS1/')
       AdminControl.testConnection(ds)
       ```

       Examples of output:

       ```
       WASX7217I: Connection to provided datasource was successful.
       ```

       ```
       DSRA0174W: Warning: GenericDataStoreHelper is being used.
       ```

       ```
       WASX7015E: Exception running command: "$AdminControl testConnection
       $ds1"; exception information:
        com.ibm.websphere.management.exception.AdminException
       javax.management.MBeanException
       java.sql.SQLRecoverableException: java.sql.SQLRecoverableException: Io
       exception: The Network Adapter could not establish the
       connectionDSRA0010E: SQL State = 08006, Error Code = 17,002
       ```

  3. Test the connection using testConnectionToDataSource.

     The following example invokes the testConnectionToDataSource operation on the MBean, passing in the classname, userid, password, database name, JDBC driver class path, language, and country.

     - Using Jacl:

       ```
       $AdminControl invoke $dshelper testConnectionToDataSource
       "COM.ibm.db2.jdbc.DB2XADataSource db2admin db2admin
       {{databaseName sample}} /sqllib/java/db2java.zip en US"
       ```

     - Using Jython:

       ```
       print AdminControl.invoke(dshelper, 'testConnectionToDataSource',
       'COM.ibm.db2.jdbc.DB2XADataSource dbuser1 dbpwd1
       "{{databaseName jtest1}}" /sqllib/java12/db \"\" \"\"')
       ```

       Example output:

       ```
       WASX7217I: Connection to provided data source was successful.
       ```

- Test the data source connection for an application scope.
  1. Start the wsadmin scripting tool.
  2. Get the data source for the application of interest.

     Use the AdminConfig object to determine the configuration IDs of the `myApplication` application and `DSA1` data source, as the following examples demonstrate:

     - Using Jacl:

       ```
       set appID [$AdminConfig getid /Deployment:myApplication/]
       set ds [$AdminConfig list DataSource $appID]
       ```

     - Using Jython:

```
appID = AdminConfig.getid("/Deployment:myApplication/")
ds = AdminConfig.list("DataSource", appID)
```

3. Test the connection using testConnection.

   Use the AdminConfig object to test the connection for the data source of interest, as the following examples demonstrate:

   – Using Jacl:

   ```
   $AdminControl testConnection $ds
   ```

   – Using Jython:

   ```
   AdminControl.testConnection(ds)
   ```

   The command returns output that indicates whether the connection is successful, as demonstrated in the following sample output:

```
WASX7467I: Connection to provided datasource on node myNode processnodeagent was successful.
WASX7217I: Connection to provided datasource was successful.
```

- Test the data source connection for a cluster scope.

  In the following example, the `Cluster1` server cluster contains cluster members on the `node1`, `node2`, and `node3` nodes. The `Cluster1` server cluster contains the `DSC1` data source.

  1. Start the wsadmin scripting tool.
  2. Get the data source configuration ID for the cluster of interest.

     Use the AdminConfig object to determine the configuration IDs of the `Cluster1` cluster and `DSA1` data source, as the following examples demonstrate:

     – Using Jacl:

     ```
     set cluster [$AdminConfig getid /ServerCluster:Cluster1/]
     set ds [$AdminConfig list DataSource $cluster]
     ```

     – Using Jython:

     ```
     cluster = AdminConfig.getid("/ServerCluster:Cluster1/")
     ds = AdminConfig.list("DataSource", cluster)
     ```

  3. Test the connection.

     Use the AdminConfig object to test the connection for the data source of interest, as the following examples demonstrate:

     – Using Jacl:

     ```
     $AdminControl testConnection $ds
     ```

     – Using Jython:

     ```
     AdminControl.testConnection(ds)
     ```

  The command returns output that indicates whether the connection is successful, as demonstrated in the following sample output:

```
WASX7467I: Connection to provided datasource on node node1 process nodeagent was successful.
WASX7467I: Connection to provided datasource on node node2 process nodeagent was successful.
WASX7467I: Connection to provided datasource on node node3 process nodeagent was successful.
WASX7217I: Connection to provided datasource was successful.
```

JDBCProviderManagement command group for AdminTask object

You can use the Jython or Jacl scripting languages in interactive mode to configure data access and data sources with scripting. The commands and parameters in the JDBCProviderManagement group can be used to create or list data sources and Java database connectivity (JDBC) providers.

The JDBCProviderManagement command group for the AdminTask object includes the following commands:

- “listDatasources” on page 39
- “listJDBCProviders” on page 39

createDatasource

The createDatasource command creates a new data source to access the back end data store. Application components use the data source to access connection instances to your database.

Target object

JDBC Provider Object ID - The configuration object of the JDBC provider to which the new data source will belong.

Required parameters

- name
The name of the data source. (String, required)

-jndiName
The Java Naming and Directory Interface (JNDI) name. (String, required)

-dataStoreHelperClassName
The name of the DataStoreHelper implementation class that extends the capabilities of the selected JDBC driver implementation class to perform data-specific functions. WebSphere Application Server provides a set of DataStoreHelper implementation classes for each of the JDBC provider drivers it supports. (String, required)

configureResourceProperties
This command step configures the resource properties that are required by the data source. (Required) You can specify the following parameters for this step:

name
The name of the resource property. (String, required)

type
The type of the resource property. (String, required)

value
The value of the resource property. (String, required)

Optional parameters

-description
The description of the data source. (String, optional)

-category
The category that you can use to classify a group of data sources. (String, optional)

-componentManagedAuthenticationAlias
The alias used for database authentication at run time. This alias is only used when the application resource reference is using res-auth=Application. (String, optional)

-containerManagedPersistence
Specifies if the data source is used for container managed persistence for enterprise beans. The default value is true. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createDatasource "DB2 Universal JDBC Driver Provider (XA)(cells/myCell|resources.xml#
JDBCProvider_1180538152781)" {-name
"DB2  Universal JDBC Driver XA DataSource" -jndiName s1 -dataStoreHelperClassName
 com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
 -componentManagedAuthenticationAlias myCellManager01/a1 -xaRecoveryAuthAlias myCellManager01/a1
 -configureResourceProperties
{{databaseName java.lang.String db1} {driverType java.lang.Integer 4} {serverName java.lang.String
dbserver1} {portNumber java.lang.Integer 50000}}}
```

- Using Jython string:

```
AdminTask.createDatasource('DB2 Universal JDBC Driver Provider(XA) (cells/myCell|resources.xml
#JDBCProvider_1180501752515)', '[-name
"DB2 Universal JDBC Driver XA DataSource 2" -jndiName ds2 -dataStoreHelperClassName
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
 -componentManagedAuthenticationAlias myCellManager01/a1 -xaRecoveryAuthAlias myCellManager01/a1
 -configureResourceProperties
[[databaseName java.lang.String db1] [driverType java.lang.Integer 4] [serverName java.lang.String
 dbserver1] [portNumber java.lang.Integer 50000]]]')
```

- Using Jython list:

```
AdminTask.createDatasource('DB2 Universal JDBC Driver Provider(XA) (cells/myCell|resources.xml#
JDBCProvider_1180501752515)', ['-name', '
DB2  Universal JDBC Driver XA DataSource 2', '-jndiName', 'ds2', '-dataStoreHelperClassName',
 'com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper', '-componentManagedAuthenticationAlias',
 'myCellManager01/a1',
 '-xaRecoveryAuthAlias', 'myCellManager01/a1', '-configureResourceProperties', '[[databaseName
java.lang.String db1]
[driverType java.lang.Integer 4] [serverName java.lang.String  dbserver1] [portNumber java.lang.Integer 50000]]']
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createDatasource {-interactive}
```

- Using Jython:

```
AdminTask.createDatasource('-interactive')
```

Note: Be aware of the following issues with the createDatasource command:

- When you create a data source that supports container-managed persistence, the createDatasource command creates two objects in the configuration of the target scope: the intended data source and a CMP connector factory that is used expressly by the container. You must apply the remove command separately to the intended data source and the CMP connector factory objects in order to delete the data source from the configuration.

  The following script illustrates how to use Jacl to remove the CMPConnectorFactory object from the configuration when you delete a data source that supports container-managed persistence:

```
# The following script removes a data source from the configuration,
# including the associated CMP connector factory
#
# Set this variable to the ID string of the data source to remove
set ds_to_remove "configID_of_my_data_source_to_remove"
puts "Data source to remove='$ds_to_remove'"

# Find and remove the CMPConectorFactory associated with the datasource to remove
foreach cmp_cf [$AdminConfig list CMPConnectorFactory] {
  set cmp_ds [lindex [$AdminConfig showAttribute $cmp_cf cmpDatasource] 0]
  if { [string compare $cmp_ds $ds_to_remove] == 0 } {
    puts "Found CMPConnectorFactory for data source $ds_to_remove"
    puts "Removing the CMPConnectorFactory '$cmp_cf'"
    $AdminConfig remove $cmp_cf
    puts "'$cmp_cf' is removed."
    break
  }
}

puts "Removing the data source '$ds_to_remove'"
$AdminConfig remove $ds_to_remove
puts "'$ds_to_remove' is removed."

#$AdminConfig save
```

- The createDatasource command will not complete successfully if you are creating a data source on a JDBC provider that does not exist or does not contain the providerType attribute. This error can be caused if you created the JDBC provider with the AdminConfig create JDBCProvider method of the JMX API. This method creates the JDBC provider, but it does not add a providerType attribute.

  The createDatasource command will issue the following error:

  ```
  "ADMF0006E: Step configureResourceProperties of command createDatasource is not found"
  ```

  To resolve this issue, use commands from the JDBCProviderManagement group, which provides the AdminTask commands createJDBCProvider and createDatasource. The AdminTask commands are designed to be used together.

  The AdminConfig JMX API methods "create JDBCProvider" and "create Datasource," which support Version 5.1 and later, are also designed to be used together. Scripts that use the JMX API methods should not use the AdminTask commands.
- The createDatasource command will not complete successfully if the JDBC provider object ID, which you specify when you create the data source, does not match the ID of the JDBC provider. If the JDBC provider object ID and the ID of the JDBC provider do not match, you can obtain the correct ID from the results of the createJDBCProvider command. Then, specify this ID when you use the createDatasource command. If you have already created the JDBC provider, use the listJDBCProviders command to list the correct ID.

createJDBCProvider

The createJDBCProvider command creates a new Java database connectivity provider (JDBC) that you can use to connect to a relational database for data access.

Target object

None

Required parameters

-scope
The scope for the new JDBC provider. Provide the scope in the form `type=name`. `Type` can be Cell, Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `none`. (String, required)

-databaseType
The type of database that will be used by the JDBC provider. For example, DB2®, Derby, Informix®, Oracle, and so on. (String, required)

-providerType
The JDBC provider type that will be used by the JDBC provider. (String, required)

-implementationType
The implementation type for this JDBC provider. Use `Connection pool data source` if your application runs in a single phase or a local transaction. Otherwise, use `XA data source` to run in a global transaction. (String, required)

Optional parameters

-classpath
Specifies a list of paths or JAR file names that form the location for the resource provider classes. (String, optional)

-description
The description for the JDBC provider. (String, optional)

-implementationClassName
Specifies the Java class name for the JDBC driver implementation. (String, optional)

-name
The name of the JDBC provider. The default is the value from the provider template. (String, optional)

-nativePath
Specifies a list of paths that form the location for the resource provider native libraries. (String, optional)

-isolated
Specifies whether the JDBC provider loads within the class loader. The default value is `false`. You cannot specify a native path for an isolated JDBC provider. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJDBCProvider {-scope Cell=my02Cell -databaseType DB2 -providerType "DB2
Universal JDBC Driver Provider"
-implementationType "XA data source" -name "DB2 Universal JDBC Driver Provider (XA)"
-description "XA DB2 Universal JDBC Driver-compliant
Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing.
Use of driver type 2 on WAS z/OS is not
supported for datasources created under this provider." -classpath
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar -nativePath ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}}
```

- Using Jython string:

```
AdminTask.createJDBCProvider('[-scope Cell=myCell -databaseType DB2 -providerType "DB2 Universal JDBC Driver Provider"
 -implementationType "XA datasource" -name "DB2 Universal JDBC Driver Provider (XA)"
-description "XA DB2 Universal JDBC
Driver-compliant Provider.
Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver
 type 2 on WAS z/OS is not supported for datasources created under this provider." -classpath
 ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar -nativePath ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}]')
```

- Using Jython list:

```
AdminTask.createJDBCProvider(['-scope', 'Cell=myCell', '-databaseType', 'DB2', '-providerType',
'DB2 Universal JDBC Driver Provider',
 '-implementationType', 'XA data source', '-name', 'DB2 Universal JDBC Driver Provider (XA)',
 '-description', 'XA DB2 Universal
JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase
commit processing. Use of
driver type 2 on WAS z/OS is not supported for datasources created under this provider.', '-classpath',
  '${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar', '-nativePath', '${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}']])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJDBCProvider {-interactive}
```

- Using Jython:

```
AdminTask.createJDBCProvider('-interactive')
```

deleteDatasource

The deleteDatasource command deletes a data source for an existing JDBC Provider at a specific scope.

Target object

DataSource Object ID - The configuration object of the DataSource to delete.

Required parameters

None.

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteDatasource "DB2 Universal JDBC Driver Provider (XA)(cells/myCell|resources.xml
#DataSource_1170538153781)"
```

- Using Jython string:

```
AdminTask.deleteDatasource('DB2 Universal JDBC Driver Provider (XA)(cells/myCell|resources.xml#
DataSource_1170538153781)')
```

Result:

The DataSource is deleted at the specific scope. Also, if DataSource is a DB2 data source configured as a Type 4 JDBC driver and enabled for Client Reroute, the JNDI name specified in either the Client reroute server list JNDI name field or the clientRerouteServerListJNDIName data source custom property is unbound.

To save the changes made by the command, invoke the save command of the AdminConfig object. If the changes are not saved, the Client reroute server list JNDI name must be rebound by issuing a test connection on DataSource or by restarting the server.

deleteJDBCProvider

The deleteJDBCProvider command deletes a JDBC provider and its data sources from a specific scope.

Target object

JDBCProvider Object ID - The configuration object of the JDBCProvider to delete.

Required parameters

None.

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteJDBCProvider "DB2 Universal JDBC Driver Provider
 (XA)(cells/myCell|resources.xml#JDBCProvider_1180538152781)"
```

- Using Jython string:

```
AdminTask.deleteJDBCProvider('DB2 Universal JDBC Driver Provider (XA)(cells/myCell|resources.xml#
JDBCProvider_1180538152781)')
```

Result:

The JDBCProvider is deleted at the specific scope. If JDBCProvider is a DB2 universal database provider, for each data source of JDBCProvider configured as a Type 4 JDBC driver and enabled for Client Reroute, the JNDI name specified in either the Client reroute server list JNDI name field or the clientRerouteServerListJNDIName data source custom property is unbound.

To save the changes made by the command, invoke the save command of the AdminConfig object. If the changes are not saved, the Client reroute server list JNDI names must be rebound by issuing test connections on the data sources of JDBCProvider or by restarting the server.

listDatasources

Use the listDatasources command to list data sources that are contained in the specified scope.

Target object

None

Required parameters

None.

Optional parameters

-scope
The scope for the data sources that will be listed. Provide the scope in the form `type=name`. `Type` can be Cell, Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `All`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listDatasources {-scope Cell=my02Cell}
```

- Using Jython string:

```
AdminTask.listDatasources('[-scope Cell=my02Cell]')
```

- Using Jython list:

```
AdminTask.listDatasources('-scope', 'Cell=my02Cell')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listDatasources {-interactive}
```

- Using Jython:

```
AdminTask.listDatasources('-interactive')
```

listJDBCProviders

The listJDBCProviders command lists JDBC providers that are contained in the specified scope.

Target object

None

Required parameters

None.

Optional parameters

-scope

The scope for the JDBC providers that will be listed. Provide the scope in the form `type=name`. `Type` can be Cell, Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `All`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJDBCProviders {-scope Cell=my02Cell}
```

- Using Jython string:

```
AdminTask.listJDBCProviders('-scope Cell=my02Cell')
```

- Using Jython list:

```
AdminTask.listJDBCProviders(['-scope', 'Cell=my02Cell'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJDBCProviders {-interactive}
```

- Using Jython:

```
AdminTask.listJDBCProviders('-interactive')
```

Chapter 3. Scripting for Mail, URLs, and other Java EE resources

This page provides a starting point for finding information about resources that are used by applications that are deployed on a Java Enterprise Edition (Java EE)-compliant application server. They include:

- JavaMail support for applications to send Internet mail
- URLs, for describing logical locations
- Resource environment entries, for mapping logical names to physical names
- Java DataBase Connectivity (JDBC) resources and other technology for data access (discussed elsewhere)
- Java Message Service (JMS) resources and other messaging system support (discussed elsewhere)

Configuring mail, URLs, and resource environment entries with wsadmin scripting

Use scripting to configure mail, URLs, and resource environment entries.

About this task

This topic contains the following tasks:

Procedure

Configuring new mail providers using wsadmin scripting

You can use scripting and the wsadmin tool to configure new mail providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new mail provider:

Procedure

1. Identify the parent ID:
   - Using Jacl:
     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```
   - Using Jython:
     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:
   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required MailProvider
     ```
   - Using Jython:
     ```
     print AdminConfig.required('MailProvider')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name MP1]
     set mpAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'MP1']
     mpAttrs = [name]
     ```
4. Create the mail provider:
   - Using Jacl:
     ```
     set newmp [$AdminConfig create MailProvider $node $mpAttrs]
     ```
   - Using Jython:
     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:
   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```
5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new mail sessions using wsadmin scripting

You can use scripting and the wsadmin tool to configure new mail sessions.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new mail session:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.getid('/Cell:mycell/Node:mynode/MailProvider:MP1/')
     print newmp
     ```

     Example output:

     ```
     MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MailSession
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MailSession')
     ```

     Example output:

     ```
     Attribute       Type
     name       String
     jndiName       String
     ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MS1]
     set jndi [list jndiName mail/MS1]
     set msAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name MS1} {jndiName mail/MS1}
     ```

   - Using Jython:

     ```
     name = ['name', 'MS1']
     jndi = ['jndiName', 'mail/MS1']
     msAttrs = [name, jndi]
     print msAttrs
     ```

     Example output:

     ```
     [[name, MS1], [jndiName, mail/MS1]]
     ```

4. Create the mail session:
   - Using Jacl:

     ```
     $AdminConfig create MailSession $newmp $msAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('MailSession', newmp, msAttrs)
     ```

     Example output:

     ```
     MS1(cells/mycell/nodes/mynode|resources.xml#MailSession_1)
     ```

5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new protocols using scripting

You can configure new protocols with scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client using wsadmin scripting” on page 549 article for more information.

About this task

Perform the following steps to configure a new protocol:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```
   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ProtocolProvider
     ```
   - Using Jython:

     ```
     print AdminConfig.required('ProtocolProvider')
     ```

   Example output:

   ```
   Attribute       Type
   protocol      String
   classname       String
   ```
3. Set up required attributes:
   - Using Jacl:

     ```
     set protocol [list protocol "Put the protocol here"]
     set classname [list classname "Put the class name here"]
     set ppAttrs [list $protocol $classname]
     ```

     Example output:

     ```
     {protocol protocol1} {classname classname1}
     ```
   - Using Jython:

     ```
     protocol = ['protocol', "Put the protocol here"]
     classname = ['classname', "Put the class name here"]
     ppAttrs = [protocol, classname]
     print ppAttrs
     ```

     Example output:

     ```
     [[protocol, protocol1], [classname, classname1]]
     ```
4. Create the protocol provider:
   - Using Jacl:

     ```
     $AdminConfig create ProtocolProvider $newmp $ppAttrs
     ```
   - Using Jython:

     ```
     print AdminConfig.create('ProtocolProvider', newmp, ppAttrs)
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#ProtocolProvider_4)
   ```
5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new custom properties using wsadmin scripting

You can use scripting and the wsadmin tool to configure new custom properties.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newmp propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newmp, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_2)
   ```

3. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute       Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name CP1]
     set cpAttrs [list $name]
     ```

     Example output:

     ```
     {name CP1}
     ```

   - Using Jython:

     ```
     name = ['name', 'CP1']
     cpAttrs = [name]
     print cpAttrs
     ```

     Example output:

     ```
     [[name, CP1]]
     ```

5. Create a J2EE resource property:

- Using Jacl:

  ```
  $AdminConfig create J2EEResourceProperty $propSet $cpAttrs
  ```

- Using Jython:

  ```
  print AdminConfig.create('J2EEResourceProperty', propSet, cpAttrs)
  ```

  Example output:

  ```
  CP1(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_2)
  ```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new resource environment providers using wsadmin scripting

You can use the wsadmin tool and scripting to configure new resources environment providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new resource environment provider:

Procedure

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ResourceEnvironmentProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ResourceEnvironmentProvider')
     ```

   Example output:

   ```
   Attribute  Type
   name     String
   ```

3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:

     ```
     set n1 [list name REP1]
     set repAttrs [list $n1]
     ```

   - Using Jython:

     ```
     n1 = ['name', 'REP1']
     repAttrs = [n1]
     ```

4. Create a new resource environment provider:
   - Using Jacl:

```
set newrep [$AdminConfig create ResourceEnvironmentProvider $node $repAttrs]
```

- Using Jython:

```
newrep = AdminConfig.create('ResourceEnvironmentProvider', node, repAttrs)
print newrep
```

Example output:

```
REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
```

5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring custom properties for resource environment providers using wsadmin scripting

You can use scripting to configure custom properties for a resource environment provider.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a resource environment provider:

Procedure

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig getid /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
     ```

   - Using Jython:

     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:

   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name     String
   ```

3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:

     ```
     set name [list name RP]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP']
     rpAttrs = [name]
     ```

4. Get the J2EE resource property set:
   - Using Jacl:

```
set propSet [$AdminConfig showAttribute $newrep propertySet]
```

- Using Jython:

```
propSet = AdminConfig.showAttribute(newrep, 'propertySet')
print propSet
```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#PropertySet_1)
```

If the command returns `None` as the value for the propSet variable, create a new property set. The command returns `None` if the property set does not exist in your environment. Use the following examples to create a new property set:

Using Jacl:

```
set newPropSet  [$AdminConfig  create  $newrep {}]
```

Using Jython:

```
newPropSet = AdminConfig.create('J2EEResourcePropertySet',newrep,[])
```

After setting the newPropSet variable, retry the command to get the J2EE resource property set before going to the next step.

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes.

   Using Jacl:

   ```
   $AdminConfig save
   ```

   Using Jython:

   ```
   AdminConfig.save()
   ```

7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new referenceables using wsadmin scripting

You can use scripting and the wsadmin tool to configure new referenceables.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new referenceable:

Procedure

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/
     ResourceEnvironmentProvider:REP1/]
     ```

   - Using Jython:

```
newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/
ResourceEnvironmentProvider:REP1/')
print newrep
```

Example output:

```
REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required Referenceable
     ```

   - Using Jython:

     ```
     print AdminConfig.required('Referenceable')
     ```

   Example output:

   ```
   Attribute     Type
   factoryClassname   String
   classname   String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set fcn [list factoryClassname REP1]
     set cn [list classname NM1]
     set refAttrs [list $fcn $cn]
     ```

   - Using Jython:

     ```
     fcn = ['factoryClassname', 'REP1']
     cn = ['classname', 'NM1']
     refAttrs = [fcn, cn]
     print refAttrs
     ```

   Example output:

   ```
   {factoryClassname {REP1}} {classname {NM1}}
   ```

4. Create a new referenceable:
   - Using Jacl:

     ```
     set newref [$AdminConfig create Referenceable  $newrep  $refAttrs]
     ```

   - Using Jython:

     ```
     newref = AdminConfig.create('Referenceable',  newrep,  refAttrs)
     print newref
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#Referenceable_1)
   ```

5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new resource environment entries using wsadmin scripting

You can use wsadmin scripting to configure a new resource environment entry.

Before you begin

Before starting this task, the wsadmin tool must be running. For more information, see “Starting the wsadmin scripting client using wsadmin scripting” on page 549. Also, you must create a resource environment provider. For more information, see the configuration topic on new resource environment providers.

About this task

Perform the following steps to configure a new resource environment entry:

Procedure

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
     ```

   - Using Jython:

     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:

   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```

2. Identify the required ResourceEnvEntry attribute:
   - Using Jacl:

     ```
     $AdminConfig required ResourceEnvEntry
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ResourceEnvEntry')
     ```

   Example output:

   ```
   Attribute   Type
   name         String
   jndiName      String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name REE1]
     set jndiName [list jndiName myjndi]
     ```

   - Using Jython:

     ```
     name = ['name', 'REE1']
     jndiName = ['jndiName', 'myjndi']
     ```

4. Identify the required Referenceable attribute:
   - Using Jacl:

     ```
     $AdminConfig required Referenceable
     ```

   - Using Jython:

     ```
     print AdminConfig.required('Referenceable')
     ```

   Example output:

   ```
   Attribute    Type
   factoryClassname String
   classname    String
   ```

5. Set up the required attributes and configure the new reference:
   - Using Jacl:

     ```
     set f1 [list factoryClassname fClass1]
     set c1 [list classname Class1]
     set refAttrs [list $f1 $c1]
     set newref [$AdminConfig create Referenceable $newrep $refAttrs]
     ```

   - Using Jython:

     ```
     f1 = ['factoryClassname', 'fClass1']
     c1 = ['classname', 'Class1']
     refAttrs = [f1, c1]
     newref = AdminConfig.create('Referenceable', newrep, refAttrs)
     print newref
     ```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#Referenceable_1)
```

6. Save the configuration changes. For more information, see the Saving configuration changes with the wsadmin tool topic.
7. In a network deployment environment only, synchronize the node. For more information, see the Synchronizing nodes with the wsadmin tool topic.

Configuring custom properties for resource environment entries using wsadmin scripting

You can use wsadmin scripting to configure a new custom property for a resource environment entry.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a resource environment entry:

Procedure

1. Identify the parent ID and assign it to the newree variable.
   - Using Jacl:

     ```
     set newree [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/]
     ```

   - Using Jython:

     ```
     newree = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/')
     print newree
     ```

   Example output:

   ```
   REE1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvEntry_1)
   ```

2. Create the J2EE custom property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newree propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newree, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_5)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP1]
     set rpAttrs [list $name]
     ```

   - Using Jython:

```
name = ['name', 'RP1']
rpAttrs = [name]
```

5. Create the J2EE custom property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RPI(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new URL providers using wsadmin scripting

You can use scripting and the wsadmin tool to configure new URL providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new URL provider:

Procedure

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required URLProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('URLProvider')
     ```

   Example output:

   ```
   Attribute     Type
   streamHandlerClassName   String
   protocol       String
   name         String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name URLP1]
     set shcn [list streamHandlerClassName "Put the stream handler classname here"]
     set protocol [list protocol "Put the protocol here"]
     set urlpAttrs [list $name $shcn $protocol]
     ```

Example output:

```
{name URLP1} {streamHandlerClassName {Put the stream handler classname here}}
{protocol {Put the protocol here}}
```

- Using Jython:

```
name = ['name', 'URLP1']
shcn = ['streamHandlerClassName', "Put the stream handler classname here"]
protocol = ['protocol', "Put the protocol here"]
urlpAttrs = [name, shcn, protocol]
print urlpAttrs
```

Example output:

```
[[name, URLP1], [streamHandlerClassName, "Put the stream handler classname here"],
[protocol, "Put the protocol here"]]
```

4. Create a URL provider:
   - Using Jacl:

     ```
     $AdminConfig create URLProvider $node $urlpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('URLProvider', node, urlpAttrs)
     ```

   Example output:

   ```
   URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
   ```

5. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring custom properties for URL providers using wsadmin

You can use the wsadmin scripting tool to configure custom properties for URL providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure custom properties for URL providers:

Procedure

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:

     ```
     set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
     ```

   - Using Jython:

     ```
     newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
     print newurlp
     ```

   Example output:

   ```
   URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newurlp propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newurlp, 'propertySet')
     print propSet
     ```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#PropertySet_7)
```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP2]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP2']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP2(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring new URLs using wsadmin scripting

You can use scripting and the wsadmin tool to configure new URLs.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following example to configure a new URL:

Procedure

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:

     ```
     set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
     ```

   - Using Jython:

     ```
     newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
     print newurlp
     ```

   Example output:

   ```
   URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required URL
     ```

   - Using Jython:

     ```
     print AdminConfig.required('URL')
     ```

   Example output:

   ```
   Attribute     Type
   name          String
   spec          String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name URL1]
     set spec [list spec "Put the spec here"]
     set urlAttrs [list $name $spec]
     ```

     Example output:

     ```
     {name URL1} {spec {Put the spec here}}
     ```

   - Using Jython:

     ```
     name = ['name', 'URL1']
     spec = ['spec', "Put the spec here"]
     urlAttrs = [name, spec]
     ```

     Example output:

     ```
     [[name, URL1], [spec, "Put the spec here"]]
     ```

4. Create a URL:
   - Using Jacl:

     ```
     $AdminConfig create URL $newurlp $urlAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('URL', newurlp, urlAttrs)
     ```

   Example output:

   ```
   URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
   ```

5. Save the configuration changes. For more information, see the documentation about saving configuration changes with the wsadmin tool.
6. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Configuring custom properties for URLs using wsadmin

Use the wsadmin scripting tool to set custom properties for URLs.

Before you begin

Before starting this task, the wsadmin tool must be running. See the topic Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a URL:

Procedure

1. Identify the parent ID and assign it to the newurl variable.
   - Using Jacl:

     ```
     set newurl [$AdminConfig getid /Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/]
     ```

- Using Jython:

```
newurl = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/')
print newurl
```

Example output:

```
URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
```

2. Create a J2EE resource property set:
   - Using Jacl:

```
set propSet [$AdminConfig showAttribute $newurl propertySet]
```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(newurl, 'propertySet')
print propSet
```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_7)
```

3. Identify the required attributes:
   - Using Jacl:

```
$AdminConfig required J2EEResourceProperty
```

   - Using Jython:

```
print AdminConfig.required('J2EEResourceProperty')
```

   Example output:

```
Attribute     Type
name         String
```

4. Set up the required attributes:
   - Using Jacl:

```
set name [list name RP3]
set rpAttrs [list $name]
```

   - Using Jython:

```
name = ['name', 'RP3']
rpAttrs = [name]
```

5. Create a J2EE resource property:
   - Using Jacl:

```
$AdminConfig create J2EEResourceProperty $propSet $rpAttrs
```

   - Using Jython:

```
print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
```

   Example output:

```
RP3(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_7)
```

6. Save the configuration changes. See the topic Saving configuration changes with the wsadmin tool for more information.
7. In a network deployment environment only, synchronize the node. See the topic Synchronizing nodes with the wsadmin tool for more information.

Provider command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure mail settings with the wsadmin tool. The commands and parameters in the provider group can be used to create, delete, and manage providers.

The Provider command group for the AdminTask object includes the following commands:

deleteProvider

The **deleteProvider** command deletes a provider from the model given the providerName.

Target object

None

Parameters and return values
- Parameters: None
- Returns: true if the provider was deleted, false if not.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask deleteProvider {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.deleteProvider ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteProvider (['-interactive'])
  ```

getProviderInfo

The **getProviderInfo** command returns a ProviderInfo object defined in the model given the providerName.

Target object

None

Parameters and return values
- Parameters: None
- Returns: A ProviderInfo object defined in the model given the providerName.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask getProviderInfo {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviderInfo ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviderInfo (['-interactive'])
  ```

getProviderInstance

The **getProviderInstance** command returns a java.security.Provider for the providerName specified.

Target object

None

Parameters and return values

- Parameters: None
- Returns: A java.security.Provider for the providerName specified.

Examples

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask getProviderInsta nce {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviderIns tance ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviderInst ance (['-interactive'])
  ```

getProviders

The **getProviders** command returns a List of ProviderInfo objects from the model.

Target object

None

Parameters and return values

- Parameters: None
- Returns: A list of ProviderInfo objects from the model.

Examples

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask getProviders {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviders ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviders (['-interactive'])
  ```

Chapter 4. Scripting for Messaging resources

This page provides a starting point for finding information about the use of asynchronous messaging resources for enterprise applications with WebSphere Application Server.

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and the Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider).
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider).
- Third-party messaging providers that implement either a JCA Version 1.5 resource adapter or the ASF component of the JMS Version 1.0.2 specification.

Configuring messaging with wsadmin scripting

Use these topics to learn about configuring messaging with the wsadmin tool. You can configure the message listener service, Java Messaging Service settings, queue and connection factories, and WebSphere MQ settings.

About this task

You can use the wsadmin tool to configure various messaging connections and settings.

Procedure

- Configure the message listener service.

  The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.
- Configure Java Messaging Service (JMS) providers, destinations, and connections.

  The application server supports asynchronous messaging through the use of a JMS provider and its related messaging system. You can use the wsadmin tool to configure new JMS providers, destinations, and connections.

  A JMS destination is used to configure the properties for the associated messaging provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging.
- Configure queue and topic connection factories for the application server. Use queue and topic connection factories to create connections between providers and destinations. A queue connection factory creates a connection to the associated JMS provider of the JMS queue destinations, for point-to-point messaging. A topic connection factory creates a connection to the associated JMS provider of JMS topic destinations, for publish and subscribe messaging.

- Configure WebSphere MQ settings. A WebSphere MQ server represents either a WebSphere MQ queue manager or a WebSphere MQ queue sharing group. It is used by service integration bus messaging to define properties used for connecting to WebSphere MQ. Setting up a WebSphere MQ server involves using the administrative console to create the server definition, add it to a service integration bus, and create a WebSphere MQ queue type destination.

Configuring resources for the default messaging provider by using the wsadmin tool

You can use these commands to manage JMS resources for the default messaging provider.

About this task

These commands provide an alternative to using the administrative console or the more complex syntax of wsadmin and JACL.

Procedure

1. Open a wsadmin command session in local mode. For example, enter `wsadmin -conntype none`.

   Note: The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts using wsadmin scripting.
2. Create the required resource by entering the appropriate command:
   - To create an activation specification use the `createSIBJMSActivationSpec` command. For more information see, createSIBJMSActivationSpec command.
   - To create a connection factory use the `createSIBJMSConnectionFactory` command. For more information see, createSIBJMSConnectionFactory command.
   - To create a queue use the `createSIBJMSQueue` command. For more information see, createSIBJMSQueue command.
   - To create a topic use the `createSIBJMSTopic` command. For more information see, createSIBJMSTopic command.

Configuring resources for WebSphere MQ messaging provider

You can use these commands to manage JMS resources for the WebSphere MQ messaging provider.

About this task

You can enter commands into a wsadmin command session to create connection factories, activation specifications, queues, and topics. These commands provide an alternative to using the administrative console or the more complex syntax of wsadmin and JACL.

Procedure

1. Open a wsadmin command session in local mode. For example, enter: `wsadmin -conntype none`.

   Note: The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts using wsadmin scripting.
2. Create the required resource by entering the appropriate command:
   - To create a connection factory, queue connection factory or topic connection factory use the `createWMQConnectionFactory` command. For more information, see createWMQConnectionFactory command.
   - To create an activation specification use the `createWMQActivationSpec` command. For more information, see:createWMQActivationSpec command.
   - To create a queue use the `createWMQQueue` command. For more information, see createWMQQueue command.

- To create a topic use the `createWMQTopic` command. For more information, see createWMQTopic command.

Configuring the message listener service by using scripting

Use scripting to configure the message listener service.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Complete the following steps to configure the message listener service for an application server:

Procedure

1. Identify the application server and assign it to the server variable:
   - Using Jacl:
     ```
     set server [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:
     ```
     server = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print server
     ```

   Example output:
   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```
2. Identify the message listener service belonging to the server and assign it to the mls variable:
   - Using Jacl:
     ```
     set mls [$AdminConfig list MessageListenerService $server]
     ```
   - Using Jython:
     ```
     mls = AdminConfig.list('MessageListenerService', server)
     print mls
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#MessageListenerService_1)
   ```
3. Modify various attributes with one of the following examples:
   - This example command changes the thread pool attributes:
     - Using Jacl:
       ```
       $AdminConfig modify $mls {{threadPool {{inactivityTimeout 4000}
       {isGrowable true} {maximumSize 100} {minimumSize 25}}}}
       ```
     - Using Jython:
       ```
       AdminConfig.modify(mls, [['threadPool', [['inactivityTimeout', 4000],
       ['isGrowable',  'true'], ['maximumSize', 100], ['minimumSize', 25]]]])
       ```
   - This example modifies the property of the first listener port:
     - Using Jacl:
       ```
       set lports [$AdminConfig showAttribute $mls listenerPorts]
       set lport [lindex $lports 0]
       $AdminConfig modify $lport {{maxRetries 2}}
       ```
     - Using Jython:
       ```
       lports = AdminConfig.showAttribute(mls, 'listenerPorts')
       cleanLports = lports[1:len(lports)-1]
       lport = cleanLports.split(" ")[0]
       AdminConfig.modify(lport, [['maxRetries', 2]])
       ```
   - This example adds a listener port:
     - Using Jacl:

```
set new [$AdminConfig create ListenerPort $mls {{name my}
{destinationJNDIName di} {connectionFactoryJNDIName jndi/fs}}]
$AdminConfig create StateManageable $new {{initialState START}}
```

– Using Jython:

```
new = AdminConfig.create('ListenerPort', mls, [['name', 'my'],
['destinationJNDIName', 'di'], ['connectionFactoryJNDIName', 'jndi/fsi']])
print new
print AdminConfig.create('stateManageable', new, [['initialState', 'START']])
```

Example output:

```
my(cells/mycell/nodes/mynode/servers/server1:server.xml#ListenerPort_1079471940692)
(cells/mycell/nodes/mynode/servers/server1:server.xml#StateManageable_107947182623)
```

4. Save the configuration changes.
5. In a network deployment environment only, synchronize the node.

Configuring new JMS providers by using scripting

You can use the wsadmin tool and scripting to configure a new Java Message Service (JMS) provider.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Perform the following steps to configure a new JMS provider:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required JMSProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('JMSProvider')
     ```

   Example output:

   ```
   Attribute                       Type
   name                            String
   externalInitialContextFactory   String
   externalProviderURL             String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name JMSP1]
     set extICF [list externalInitialContextFactory
     "Put the external initial context factory here"]
     set extPURL [list externalProviderURL "Put the external provider URL here"]
     set jmspAttrs [list $name $extICF $extPURL]
     ```

   - Using Jython:

```
name = ['name', 'JMSP1']
extICF = ['externalInitialContextFactory',
"Put the external initial context factory here"]
extPURL = ['externalProviderURL', "Put the external provider URL here"]
jmspAttrs = [name, extICF, extPURL]
print jmspAttrs
```

Example output:

```
{name JMSP1} {externalInitialContextFactory {Put the external
initial context factory here }} {externalProviderURL
{Put the external provider URL here}}
```

4. Create the JMS provider:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig create JMSProvider $node $jmspAttrs]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.create('JMSProvider', node, jmspAttrs)
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new JMS destinations by using scripting

You can use scripting and the wsadmin tool to configure a new Java Message Service (JMS) destination.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Perform the following steps to configure a new JMS destination:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required GenericJMSDestination
     ```

   - Using Jython:

     ```
     print AdminConfig.required('GenericJMSDestination')
     ```

   Example output:

   ```
   Attribute          Type
   name               String
   jndiName           String
   externalJNDIName   String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name JMSD1]
     set jndi [list jndiName jms/JMSDestination1]
     set extJndi [list externalJNDIName jms/extJMSD1]
     set jmsdAttrs [list $name $jndi $extJndi]
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSD1']
     jndi = ['jndiName', 'jms/JMSDestination1']
     extJndi = ['externalJNDIName', 'jms/extJMSD1']
     jmsdAttrs = [name, jndi, extJndi]
     print jmsdAttrs
     ```

   Example output:

   ```
   {name JMSD1} {jndiName jms/JMSDestination1} {externalJNDIName jms/extJMSD1}
   ```

4. Create the generic JMS destination:
   - Using Jacl:

     ```
     $AdminConfig create GenericJMSDestination $newjmsp  $jmsdAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('GenericJMSDestination', newjmsp,  jmsdAttrs)
     ```

   Example output:

   ```
   JMSD1(cells/mycell/nodes/mynode|resources.xml#GenericJMSDestination_1)
   ```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new JMS connections by using wsadmin scripting

Use the wsadmin scripting tool to configure a new Java Message Service (JMS) connection.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Perform the following steps to configure a new JMS connection:

Procedure

1. Configure a connection pool for your generic connection factories.

   Because Java 2 Connector (J2C) manages the generic connection factories, you must configure a connection pool to indicate the policy for connection management by J2C. The following example commands configure a connection pool in your environment:
   - Using Jacl:

     ```
     set connectionPool [$AdminConfig create ConnectionPool $yourGenericCF {} connectionPool]
     set sessionPool [$AdminConfig create ConnectionPool $yourGenericCF {} sessionPool]
     ```

   - Using Jython:

     ```
     connectionPool = AdminConfig.create('ConnectionPool', '[yourGenericCF {}, connectionPool]')
     sessionPool = AdminConfig.create('ConnectionPool', '[yourGenericCF {}, sessionPool]')
     ```

2. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```

   - Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
print newjmsp
```

Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

3. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required GenericJMSConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('GenericJMSConnectionFactory')
     ```

   Example output:

   ```
   Attribute          Type
   name               String
   jndiName           String
   externalJNDIName   String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name JMSCF1]
     set jndi [list jndiName jms/JMSConnFact1]
     set extJndi [list externalJNDIName jms/extJMSCF1]
     set jmscfAttrs [list $name $jndi $extJndi]
     ```

     Example output:

     ```
     {name JMSCF1} {jndiName jms/JMSConnFact1} {externalJNDIName jms/extJMSCF1}
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSCF1']
     jndi = ['jndiName', 'jms/JMSConnFact1']
     extJndi = ['externalJNDIName', 'jms/extJMSCF1']
     jmscfAttrs = [name, jndi, extJndi]
     print jmscfAttrs
     ```

     Example output:

     ```
     [[name, JMSCF1], [jndiName, jms/JMSConnFact1], [externalJNDIName, jms/extJMSCF1]]
     ```

5. Create the generic JMS connection factory:
   - Using Jacl:

     ```
     $AdminConfig create GenericJMSConnectionFactory $newjmsp $jmscfAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('GenericJMSConnectionFactory', newjmsp, jmscfAttrs)
     ```

   Example output:

   ```
   JMSCF1(cells/mycell/nodes/mynode|resources.xml#GenericJMSConnectionFactory_1)
   ```

6. Save the configuration changes.
7. In a network deployment environment only, synchronize the node.

Configuring new queue connection factories by using scripting

You can use scripting and the wsadmin tool to configure new queue connection factories in WebSphere Application Server.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Perform the following steps to configure a new queue connection factory:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:

   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASQueueConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASQueueConnectionFactory')
     ```

   Example output:

   ```
   Attribute       Type
   name            String
   jndiName        String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name WASQCF]
     set jndi [list jndiName jms/WASQCF]
     set mqcfAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name WASQCF} {jndiName jms/WASQCF}
     ```

   - Using Jython:

     ```
     name = ['name', 'WASQCF']
     jndi = ['jndiName', 'jms/WASQCF']
     mqcfAttrs = [name, jndi]
     print mqcfAttrs
     ```

     Example output:

     ```
     [[name, WASQCF], [jndiName, jms/WASQCF]]
     ```

4. Set up a template:
   - Using Jacl:

     ```
     set template [lindex [$AdminConfig listTemplates WASQueueConnectionFactory] 0]
     ```

   - Using Jython:

     ```
     lineseparator = java.lang.System.getProperty('line.separator')
     template = AdminConfig.listTemplates('WASQueueConnectionFactory').split(lineseparator)[0]
     print template
     ```

5. Create the queue connection factory:
   - Using Jacl:

     ```
     $AdminConfig createUsingTemplate WASQueueConnectionFactory $v5jmsp $mqcfAttrs $template
     ```

   - Using Jython:

     ```
     AdminConfig.createUsingTemplate('WASQueueConnectionFactory', v5jmsp, mqqcfAttrs, template)
     ```

   Example output:

   ```
   WASQCF(cells/mycell/nodes/mynode|resources.xml#WASQueueConnectionFactory_1)
   ```

6. Save the configuration changes.
7. In a network deployment environment only, synchronize the node.

Configuring new topic connection factories by using scripting

Use scripting and the wsadmin tool to configure new topic connection factories.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Complete the following steps to configure a new topic connection factory:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

     Example output:

     ```
     "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
     ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASTopicConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASTopicConnectionFactory')
     ```

     Example output:

     ```
     Attribute       Type
     name            String
     jndiName        String
     port            ENUM(DIRECT, QUEUED)
     ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name WASTCF]
     set jndi [list jndiName jms/WASTCF]
     set port [list port QUEUED]
     set mtcfAttrs [list $name $jndi $port]
     ```

     Example output:

     ```
     {name WASTCF} {jndiName jms/WASTCF} {port QUEUED}
     ```

   - Using Jython:

     ```
     name = ['name', 'WASTCF']
     jndi = ['jndiName', 'jms/WASTCF']
     port = ['port', 'QUEUED']
     mtcfAttrs = [name, jndi, port]
     print mtcfAttrs
     ```

     Example output:

     ```
     [[name, WASTCF], [jndiName, jms/WASTCF], [port, QUEUED]]
     ```

4. Create was topic connection factories:
   - Using Jacl:

     ```
     $AdminConfig create WASTopicConnectionFactory $v5jmsp $mtcfAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WASTopicConnectionFactory', v5jmsp, mtcfAttrs)
     ```

Example output:

```
WASTCF(cells/mycell/nodes/mynode|resources.xml#WASTopicConnectionFactory_1)
```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new queues by using scripting

You can use scripting to configure a new queue.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Complete the following steps to configure a new queue:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:

   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASQueue
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASQueue')
     ```

   Example output:

   ```
   Attribute       Type
   name            String
   jndiName        String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name WASQ1]
     set jndi [list jndiName jms/WASQ1]
     set wqAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name WASQ1} {jndiName jms/WASQ1}
     ```

   - Using Jython:

     ```
     name = ['name', 'WASQ1']
     jndi = ['jndiName', 'jms/WASQ1']
     wqAttrs = [name, jndi]
     print wqAttrs
     ```

     Example output:

     ```
     [[name, WASQ1], [jndiName, jms/WASQ1]]
     ```

4. Create the queue:
   - Using Jacl:

```
$AdminConfig create WASQueue $v5jmsp $wqAttrs
```

- Using Jython:

```
print AdminConfig.create('WASQueue', v5jmsp, wqAttrs)
```

Example output:

```
WASQ1(cells/mycell/nodes/mynode|resources.xml#WASQueue_1)
```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

Configuring new topics by using scripting

You can configure new topics by using the wsadmin tool and scripting.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Complete the following steps to configure a new topic:

Procedure

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:

   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASTopic
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASTopic')
     ```

   Example output:

   ```
   Attribute      Type
   name           String
   jndiName       String
   topic          String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name WAST1]
     set jndi [list jndiName jms/WAST1]
     set topic [list topic "Put your topic here"]
     set wtAttrs [list $name $jndi $topic]
     ```

     Example output:

     ```
     {name WAST1} {jndiName jms/WAST1} {topic {Put your topic here}}
     ```

   - Using Jython:

```
name = ['name', 'WAST1']
jndi = ['jndiName', 'jms/WAST1']
topic = ['topic', "Put your topic here"]
wtAttrs = [name, jndi, topic]
print wtAttrs
```

Example output:

```
[[name, WAST1], [jndiName, jms/WAST1], [topic, "Put your topic here"]]
```

4. Create the topic:
   - Using Jacl:

     ```
     $AdminConfig create WASTopic $v5jmsp $wtAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WASTopic', v5jmsp, wtAttrs)
     ```

   Example output:

   ```
   WAST1(cells/mycell/nodes/mynode|resources.xml#WASTopic_1)
   ```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.

JCAManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure messaging with scripting. The commands and parameters in the JCA management group can be used to create and manage resource adapters, Java EE Connector Architecture (J2C) activation specifications, administrative objects, connection factories, administrative object interfaces, and message listener types.

The JCAManagement command group for the AdminTask object includes the following commands:

copyResourceAdapter

Use the copyResourceAdapter command to create a Java 2 Connector (J2C) resource adapter under the scope that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-name
Indicates the name of the new J2C resource adapter. This parameter is required.

-scope
Indicates the scope object ID. This parameter is required.

-useDeepCopy
If you set this parameter to `true`, all of the J2C connection factory, J2C activation specification, and J2C administrative objects will be copied to the new J2C resource adapter (deep copy). If you set this parameter to `false`, the objects are not created (shallow copy). The default is `false`.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask copyResourceAdapter $ra [subst {-name newRA -scope $scope}]
```

- Using Jython string:

```
AdminTask.copyResourceAdapter(ra, '[-name newRA -scope scope]')
```

- Using Jython list:

```
AdminTask.copyResourceAdapter(ra, ['-name', 'newRA', '-scope', 'scope'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask copyResourceAdapter {-interactive}
```

- Using Jython:

```
AdminTask.copyResourceAdapter('-interactive')
```

createJ2CActivationSpec

Use the createJ2CActivationSpec command to create a Java 2 Connector (J2C) activation specification under a J2C resource adapter and the attributes that you specify. Use the messageListenerType parameter to indicate the activation specification that is defined for the J2C resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

- messageListenerType
Identifies the activation specification for the J2C activation specification to be created. Use this parameter to identify the activation specification template for the J2C resource adapter that you specify.

-name
Indicates the name of the J2C activation specification that you are creating.

-jndiName
Indicates the name of the Java Naming and Directory Interface (JNDI).

-destinationJndiName
Indicates the name of the Java Naming and Directory Interface (JNDI) of corresponding destination.

-authenticationAlias
Indicates the authentication alias of the J2C activation specification that you are creating.

-description
Description of the created J2C activation specification.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CActivationSpec $ra {-name J2CAct Spec -jndiName eis/ActSpec1
 -messageListenerType javax.jms.MessageListener }
```

- Using Jython string:

```
AdminTask.createJ2CActivationSpec(ra, '[-name J2CActSpec -jndiName eis/ActSpec1
 -messageListenerType javax.jms.MessageListener]')
```

- Using Jython list:

```
AdminTask.createJ2CActivationSpec(ra, ['-name', 'J2CActSpec', '-jndiName', 'eis/ActSpec1',
 '-messageListenerType', 'javax.jms.MessageListener'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CActivationSpec {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CActivationSpec('-interactive')
```

createJ2CAdminObject

Use the createJ2CAdminObject command to create an administrative object under a resource adapter with attributes that you specify. Use the administrative object interface to indicate the administrative object that is defined in the resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

-adminObjectInterface
Specifies the administrative object interface to identify the administrative object for the resource adapter that you specify. This parameter is required.

-name
Indicates the name of the administrative object.

-jndiName
Specifies the name of the Java Naming and Directory Interface (JNDI).

-description
Description of the created J2C admin object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CAdminObject $ra {-adminObjectInterface fvt.adapter.message.FVTMessageProvider
 -name J2CA01 -jndiName eis/J2CA01}
```

- Using Jython string:

```
AdminTask.createJ2CAdminObject(ra, '[-adminObjectInterface fvt.adapter.message.FVTMessageProvider
 -name J2CA01 -jndiName eis/J2CA01]')
```

- Using Jython list:

```
AdminTask.createJ2CAdminObject(ra, ['-adminObjectInterface',
 'fvt.adapter.message.FVTMessageProvider', '-name', 'J2CA01', '-jndiName', 'eis/J2CA01'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CAdminObject {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CAdminObject('-interactive')
```

createJ2CConnectionFactory

Use the createJ2CConnectionFactory command to create a Java 2 connection factory under a Java 2 resource adapter and the attributes that you specify. Use the connection factory interfaces to indicate the connection definitions that are defined for the Java 2 resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

-connectionFactoryInterface
Identifies the connection definition for the Java 2 resource adapter that you specify. This parameter is required.

-name
Indicates the name of the connection factory.

-jndiName
Indicates the name of the Java Naming and Directory Interface (JNDI).

-description
Description of the created J2C connection factory.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CConnectionFactory $ra {-connectionFactoryInterfaces javax.sql.DataSource
-name J2CCF1 -jndiName eis/J2CCF1}
```

- Using Jython string:

```
AdminTask.createJ2CConnectionFactory(ra, '[-connectionFactoryInterfaces javax.sql.DataSource
-name J2CCF1 -jndi Name eis/J2CCF1]')
```

- Using Jython list:

```
AdminTask.createJ2CConnectionFactory(ra, ['-connectionFactoryInterfaces',
 'javax.sql.DataSource', '-name', 'J2CCF1', '-jndiName', 'eis/J2CCF1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CConnectionFactory {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CConnectionFactory('-interactive')
```

listAdminObjectInterfaces

Use the listAdminObjectInterfaces command to list the administrative object interfaces that are defined under the resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None

- Returns: A list of administrative object interfaces.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAdminObjectInterfaces $ra
```

- Using Jython string:

```
AdminTask.listAdminObjectInterfaces(ra)
```

- Using Jython list:

```
AdminTask.listAdminObjectInterfaces(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAdminObjectInterfaces {-interactive}
```

- Using Jython:

```
AdminTask.listAdminObjectInterfaces('-interactive')
```

listConnectionFactoryInterfaces

Use the listConnectionFactoryInterfaces command to list all of the connection factory interfaces that are defined under the Java 2 connector resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None
- Returns: A list of connection factory interfaces.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listConnectionFactoryInterfaces $ra
```

- Using Jython string:

```
AdminTask.listConnectionFactoryInterfaces(ra)
```

- Using Jython list:

```
AdminTask.listConnectionFactoryInterfaces(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listConnectionFactoryInterfaces {-interactive}
```

- Using Jython:

```
AdminTask.listConnectionFactoryInterfaces('-interactive')
```

listJ2CActivationSpecs

Use the listJ2CActivationSpecs command to list the activation specifications that are contained under the resource adapter and message listener type that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-messageListenerType
Specifies the message listener type for the resource adapter for which you are making a list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CActivationSpecs $ra {-messageListener Type javax.jms.MessageListener}
```

- Using Jython string:

```
AdminTask.listJ2CActivationSpecs(ra, '[-messageListener Type javax.jms.MessageListener]')
```

- Using Jython list:

```
AdminTask.listJ2CActivation Specs(ra, ['-messageListener Type', 'javax.jms.MessageListener'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CActivationSpecs {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CActivationSpecs('-interactive')
```

listJ2CAdminObjects

Use the listJ2CAdminObjects command to list administrative objects that contain the administrative object interface that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-adminObjectInterface
Specifies the administrative object interface for which you want to list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CAdminObjects $ra {-adminObjectInterface
 fvt.adaptor.message.FVTMessageProvider}
```

- Using Jython string:

```
AdminTask.listJ2CAdminObjects(ra, '[-adminObjectInterface
 fvt.adaptor.message.FVTMessageProvider]')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CAdminObjects {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CAdminObjects('-interactive')
```

listJ2CConnectionFactories

Use the listJ2CConnectionFactories command to list the Java 2 connector connection factories under the resource adapter and connection factory interface that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-connectionFactoryInterface
Indicates the name of the connection factory that you want to list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CConnectionFactories $ra {-connectionFactoryInterface javax.sql.DataSource}
```

- Using Jython string:

```
AdminTask.listJ2CConnectionFactories(ra, '[-connectionFactoryInterface javax.sql.DataSource]')
```

- Using Jython list:

```
AdminTask.listJ2CConnectionFactories(ra, ['-connectionFactoryInterface',
 'javax.sql.DataSource'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CConnectionFactories {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CConnectionFactories('-interactive')
```

listMessageListenerTypes

Use the listMessageListenerTypes command to list the message listener types that are defined under the resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None
- Returns: A list of message listener types.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listMessageListenerTypes $ra
```

- Using Jython string:

```
AdminTask.listMessageListenerTypes(ra)
```

- Using Jython list:

```
AdminTask.listMessageListenerTypes(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listMessageListenerTypes {-interactive}
```

- Using Jython:

```
AdminTask.listMessageListenerTypes('-interactive')
```

Chapter 5. Scripting for Naming and directory

This page provides a starting point for finding information about naming support. Naming includes both server-side and client-side components. The server-side component is a Common Object Request Broker Architecture (CORBA) naming service (CosNaming). The client-side component is a Java™ Naming and Directory Interface (JNDI) service provider. JNDI is a core component in the Java Platform, Enterprise Edition (Java EE) programming model.

The WebSphere® JNDI service provider can be used to interoperate with any CosNaming name server implementation. Yet WebSphere name servers implement an extension to CosNaming, and the JNDI service provider uses those WebSphere extensions to provide greater capability than CosNaming alone. Some added capabilities are binding and looking up of non-CORBA objects.

Java EE applications use the JNDI service provider supported by WebSphere Application Server to obtain references to objects related to server applications, such as enterprise bean (EJB) homes, which have been bound into a CosNaming name space.

Configuring namespace bindings using the wsadmin scripting tool

Use this topic to configure name space bindings with the Jython or Jacl scripting languages and the wsadmin tool.

Before you begin

About this task

Use this task and the following examples to configure string, Enterprise JavaBeans (EJB), CORBA, or indirect name space bindings on a cell.

Procedure

1. Start the wsadmin scripting tool.
2. Identify the cell and assign it to the cell variable.

   Using Jacl:

```
set cell [$AdminConfig getid /Cell:mycell/]
```

   Example output:

```
mycell(cells/mycell|cell.xml#Cell_1)
```

   Using Jython:

```
cell = AdminConfig.getid('/Cell:mycell/')
print cell
```

   You can change this example to configure on a node or server here.
3. Add a new name space binding on the cell. There are four binding types to choose from when configuring a new name space binding. They are string, EJB, CORBA, and indirect.
   - To configure a string type name space binding:

     Using Jacl:

```
$AdminConfig create StringNameSpaceBinding $cell {{name binding1} {nameInNameSpace
myBindings/myString} {stringToBind "This is the String value that gets bound"}}
```

     Example output:

```
binding1(cells/mycell|namebindings.xml#StringNameSpaceBinding_1)
```

     Using Jython:

```
print AdminConfig.create('StringNameSpaceBinding', cell, [['name', 'binding1'],
['nameInNameSpace', 'myBindings/myString'], ['stringToBind', "This is the String value that gets bound"]])
```

   - To configure an EJB type name space binding:

Using Jacl:

```
$AdminConfig create EjbNameSpaceBinding $cell {{name binding2} {nameInNameSpace myBindings/myEJB}
{applicationNodeName mynode} {bindingLocation SINGLESERVER} {applicationServerName server1}
{ejbJndiName ejb/myEJB}}
```

Using Jython:

```
print AdminConfig.create('EjbNameSpaceBinding', cell, [['name', 'binding2'],  ['nameInNameSpace',
'myBindings/myEJB'], ['applicationNodeName', 'mynode'],  ['bindingLocation', 'SINGLESERVER'],
['applicationServerName', 'server1'], ['ejbJndiName', 'ejb/myEJB']])
```

This example is for an EJB located in a server. For an EJB in a cluster, change the configuration example to:

Using Jacl:

```
$AdminConfig create EjbNameSpaceBinding $cell {{name binding2} {nameInNameSpace myBindings/myEJB}
{bindingLocation SERVERCLUSTER} {applicationServerName cluster1} {ejbJndiName ejb/myEJB}}
```

Using Jython:

```
print AdminConfig.create('EjbNameSpaceBinding', cell, [['name','binding2'],
['nameInNameSpace','myBindings/myEJB'], ['bindingLocation','SERVERCLUSTER'],
['applicationServerName','cluster1'], ['ejbJndiName','ejb/myEJB']])
```

Example output:

```
binding2(cells/mycell|namebindings.xml#EjbNameSpaceBinding_1)
```

- To configure a CORBA type name space binding:

  Using Jacl:

```
$AdminConfig create CORBAObjectNameSpaceBinding $cell {{name binding3} {nameInNameSpace
myBindings/myCORBA}{corbanameUrl corbaname:iiop:somehost.somecompany.com:2809#stuff/MyCORBAObject}
{federatedContext false}}
```

  Example output:

```
binding3(cells/mycell|namebindings.xml#CORBAObjectNameSpaceBinding_1)
```

  Using Jython:

```
print AdminConfig.create('CORBAObjectNameSpaceBinding', cell, [['name', 'binding3'],['nameInNameSpace',
'myBindings/myCORBA'],['corbanameUrl','corbaname:iiop:somehost.somecompany.com:2809#stuff/MyCORBAObject'],
['federatedContext', 'false']])
```

- To configure an indirect type name space binding:

  Using Jacl:

```
$AdminConfig create IndirectLookupNameSpaceBinding $cell
{{name binding4} {nameInNameSpace myBindings/myIndirect} {providerURL
corbaloc::myCompany.com:9809/NameServiceServerRoot} {jndiName jndi/name/for/EJB}}
```

  Example output:

```
binding4(cells/mycell|namebindings.xml#IndirectLookupNameSpaceBinding_1)
```

  Using Jython:

```
print AdminConfig.create('IndirectLookupNameSpaceBinding', cell, [['name', 'binding4'],
['nameInNameSpace','myBindings/myIndirect'], ['providerURL', 'corbaloc::myCompany.com:9809/NameServiceServerRoot'],
['jndiName', 'jndi/name/for/EJB']])
```

4. Save the configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Chapter 6. Scripting for security

Configuring security with scripting

You can configure security with scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. Read about Starting the wsadmin scripting client for more information.

About this task

If you enable security for an application server cell, supply authentication information to communicate with servers. The `sas.client.props` and the `soap.client.props` files are located in the following properties directory for each application server profile:

- *profile_root*`/properties`

Procedure

- The nature of the properties file updates required for running in secure mode depend on whether you connect with a Remote Method Invocation (RMI) connector, a JSR160RMI connector, an Inter-Process Communications (IPC) or a SOAP connector:
  - If you use a Remote Method Invocation (RMI) connector or a JSR160RMI connector, set the following properties in the `sas.client.props` file with the appropriate values:

```
com.ibm.CORBA.loginUserid=
com.ibm.CORBA.loginPassword=
```

    Also, set the following property:

```
com.ibm.CORBA.loginSource=properties
```

    The default value for this property is `prompt` in the `sas.client.props` file. If you leave the default value, then a dialog box is displayed with a password prompt. If the script is running unattended, then the system stops.
  - If you use a SOAP connector, set the following properties in the `soap.client.props` file with the appropriate values:

```
com.ibm.SOAP.securityEnabled=true
com.ibm.SOAP.loginUserid=
com.ibm.SOAP.loginPassword=
```

  - If you use an IPC connector, set the following properties in the `ipc.client.props` file with the appropriate values:

```
com.ibm.IPC.loginUserid=
com.ibm.IPC.loginPassword=
```

- Specify user and password information. Choose one of the following methods:
  - Specify user name and password on a command line, using the **-user** and **-password** commands, as the following examples demonstrate:

```
wsadmin -conntype JSR160RMI -port 2809 -user u1 -password secret1
```

  - Specify user name and password in the `sas.client.props` file for an RMI connector, the `ipc.client.props` file for the IPC connector, or the `soap.client.props` file for a SOAP connector.

  If you specify user and password information on a command line and in the `sas.client.props` file or the `soap.client.props` file, the command line information overrides the information in the `props` file.

Enabling and disabling security using scripting

You can use scripting to enable or disable application security, global security, administrative security based on the LocalOS registry, and authentication mechanisms.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

The default profile sets up procedures so that you can enable and disable administrative security based on LocalOS registry.

Procedure

- Use the isAppEnabled command to determine if application security is enabled or disabled, as the following example demonstrates:
  - Using Jacl:

```
$AdminTask isAppSecurityEnabled {}
```

  - Using Jython:

```
AdminTask.isAppSecurityEnabled()
```

  This command returns a value of `true` if appEnabled is set to `true`. Otherwise, returns a value of `false`.
- Use the isGlobalSecurityEnabled command to determine if administrative security is enabled or disabled, as the following example demonstrates:
  - Using Jacl:

```
$AdminTask isGlobalSecurityEnabled{}
```

  - Using Jython:

```
AdminTask.isGlobalSecurityEnabled()
```

  Returns a value of `true` if enabled is set to `true`. Otherwise, returns a value of `false`.
- Use the setGlobalSecurity command to set administrative security based on the passed in value, as the following example demonstrates:
  - Using Jacl:

```
$AdminTask setGlobalSecurity {-enabled true}
```

  - Using Jython:

```
AdminTask.setGlobalSecurity ('[-enabled true]')
```

  Returns a value of `true` if the enabled field in the WCCM security model is successfully updated. Otherwise, returns a value of `false`.
- Use the **help** command to find out the arguments that you need to provide with this call, as the following example demonstrates:
  - Using Jacl:

```
securityon help
```

    Example output:

```
Syntax: securityon user password
```

  - Using Jython:

```
securityon()
```

    Example output:

```
Syntax: securityon(user, password)
```

- Enable administrative security based on the LocalOS registry, as the following example demonstrates:
  - Using Jacl:

```
securityon user1 password1
```

  - Using Jython:

```
securityon('user1', 'password1')
```

- Disable administrative security based on the LocalOS registry, as the following example demonstrates:
  - Using Jacl:

```
securityoff
```

– Using Jython:

```
securityoff()
```

- Enable and disable LTPA and Kerberos authentication.

  Use the setActiveAuthMechanism command to set Kerberos as the authentication mechanism in the security configuration, as the following example demonstrates:

```
AdminTask.setActiveAuthMechanism('-authMechanismType KRB5')
```

  Use the setActiveAuthMechanism command to set LTPA as the authentication mechanism in the security configuration, as the following example demonstrates:

```
AdminTask.setActiveAuthMechanism('-authMechanismType LTPA')
```

  Additionally, there are sample scripts located in the <WAS_ROOT>/bin directory on how to enable and disable LTPA authentication. The scripts are:

  – LTPA_LDAPSecurityProcs.py (python script)
  – LTPA_LDAPSecurityProcs.jacl (jacl script)

  Note: The scripts hard code the type of LDAP server and base distinguished name (baseDN). The LDAP server type is hardcoded as IBM_DIRECTORY_SERVER and the baseDN is hardcoded as o=ibm,cn=us.

Enabling and disabling Java 2 security using scripting

You can enable or disable Java 2 security with scripting and the wsadmin tool.

About this task

There are two ways to enable or disable Java 2 security. You can use the commands for the AdminConfig object, or you can use the setAdminActiveSecuritySettings command for the AdminTask object.

Procedure

1. Use the setAdminActiveSecuritySettings command for the AdminTask object to enable or disable Java 2 security.
   a. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
   b. Use the getActiveSecuritySettings command to display the current security settings, including custom properties for global security, as the following example demonstrates:
      - Using Jacl:

```
$AdminTask getActiveSecuritySettings
```

      - Using Jython:

```
AdminTask.getActiveSecuritySettings()
```

   c. Use the setAdminActiveSecuritySettings command to enable or disable Java 2 security.

      The following examples enable Java 2 security:

      - Using Jacl:

```
$AdminTask setAdminActiveSecuritySettings {-enforceJava2Security true}
```

      - Using Jython:

```
AdminTask.setAdminActiveSecuritySettings('-enforceJava2Security true')
```

      The following examples disable Java 2 security:

      - Using Jacl:

```
$AdminTask setAdminActiveSecuritySettings {-enforceJava2Security false}
```

      - Using Jython:

```
AdminTask.setAdminActiveSecuritySettings('-enforceJava2Security false')
```

   d. Save the configuration changes.

      Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

e. Synchronize the node.

Use the syncActiveNode or syncNode scripts in the AdminNodeManagement script library to propagate the configuration changes to node or nodes.

- Use the syncActiveNodes script to propagate the changes to each node in the cell, as the following example demonstrates:

```
AdminNodeManagement.syncActiveNodes()
```

- Use the syncNode script to propagate the changes to a specific node, as the following example demonstrates:

```
AdminNodeManagement.syncNode("myNode")
```

2. Use the AdminConfig object to enable Java 2 security.
   a. Start the wsadmin scripting tool.
   b. Identify the security configuration object and assign it to the security variable, as the following example demonstrates:
      - Using Jacl:

```
set security [$AdminConfig list Security]
```

      - Using Jython:

```
security = AdminConfig.list('Security')
print security
```

Example output:

```
(cells/mycell|security.xml#Security_1)
```

   c. Modify the enforceJava2Security attribute to enable or disable Java 2 security, as the following examples demonstrates:
      - To enable Java 2 security:
        - Using Jacl:

```
$AdminConfig modify $security {{enforceJava2Security true}}
```

        - Using Jython:

```
AdminConfig.modify(security, [['enforceJava2Security', 'true']])
```

      - To disable Java 2 security:
        - Using Jacl:

```
$AdminConfig modify $security {{enforceJava2Security false}}
```

        - Using Jython:

```
AdminConfig.modify(security, [['enforceJava2Security', 'false']])
```

   d. Save the configuration changes.

Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

   e. Synchronize the node.

Use the syncActiveNode or syncNode scripts in the AdminNodeManagement script library to propagate the configuration changes to node or nodes.

- Use the syncActiveNodes script to propagate the changes to each node in the cell, as the following example demonstrates:

```
AdminNodeManagement.syncActiveNodes()
```

- Use the syncNode script to propagate the changes to a specific node, as the following example demonstrates:

```
AdminNodeManagement.syncNode("myNode")
```

WizardCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the WizardCommands group can be used to configure security using similar actions to the security wizard panels in the administrative console.

The WizardCommands command group for the AdminTask object includes the following commands:

addToAdminAuthz

The **addToAdminAuthz** command adds a new administrative user to your configuration.

Required parameters

adminUser
Specifies the name of the administrative user that you want to add to your configuration.

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask addToAdminAuthz {-adminUser user_name}
  ```
- Using Jython string:
  ```
  AdminTask.addToAdminAuthz ('[-adminUser user_name]')
  ```
- Using Jython list:
  ```
  AdminTask.addToAdminAuthz (['-adminUser', 'user_name'])
  ```

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask addToAdminAuthz {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.addToAdminAuthz ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.addToAdminAuthz (['-interactive'])
  ```

applyWizardSettings

The **applyWizardSettings** command applies the current security wizard settings from the workspace.

Required parameters

adminName
Specifies the name of the user with administrative privileges that is defined in the registry.

secureApps
Specifies whether to set application-level security. This type of security provides application isolation and requirements for authenticating application users.

You can specify a true or false value.

Note: The value that you set for this parameter might be overridden by a value at the server level.

secureLocalResources
Specifies whether to set Java 2 security. If you enable Java 2 security and an application requires more Java 2 security permissions than are granted in the default policy, then the application might fail to run properly. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

You can specify a true or false value.

userRegistryType
Specifies a valid user registry type. The following type values are valid:

- `LDAPUserRegistry`

  This registry type uses the Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups exist in an external LDAP directory.
- `CustomUserRegistry`

  This type specifies a custom registry that implements the UserRegistry interface in the com.ibm.websphere.security package. If you specify this user registry type, use the **customRegistryClass** parameter to specify the class name for the user registry.
- `WIMUserRegistry`

  This value has the same effect as the Federated repositories option in the Security Configuration Wizard on the administrative console. A registry type manages identities in a single, virtual realm that is stored in multiple repositories.
- `LocalOSUserRegistry`

  This value specifies the registry for the local operating system of the application server.

Optional parameters

adminPassword
Specifies a password for the user with administrative privileges that is defined in the registry.

customProps
Specifies a custom property.

customRegistryClass
Specifies a dot-separated class name that implements the UserRegistry interface in the com.ibm.websphere.security package. Include this parameter if you specify `CustomUserRegistry` for the **userRegistryType** parameter.

ignoreCase
Indicates that when an authorization check is performed, the check is not case-sensitive.

You can specify a true or false value.

ldapServerType
Specifies a valid Lightweight Directory Access Protocol (LDAP) server type. The following type values are valid:

- `IBM_DIRECTORY_SERVER`

  This value refers to a supported IBM Tivoli® Directory Server version.
- `IPLANET`

  This value refers to a supported Sun Java System Directory Server version.
- `NDS`

  This value refers to a supported Novell eDirectory version.
- `DOMINO502`

This value refers to a supported IBM Lotus® Domino® server version.

- SECUREWAY

  This value refers to an IBM SecureWay™ Directory Server version.

- ACTIVE_DIRECTORY

  This value refers to a supported Microsoft Active Directory version.

- CUSTOM

  This value refers to a custom registry implementation.

For more information about the supported LDAP server versions, see the WebSphere Application Server detailed system requirements documentation.

ldapBaseDN
Specifies the base distinguished name of the directory service, which indicates the starting point for Lightweight Directory Access Protocol (LDAP) searches in the directory service. For example, ou=Rochester, o=IBM, c=us.

ldapBindDN
Specifies the distinguished name for the application server, which is used to bind to the directory service.

ldapBindPassword
Specifies the password for the application server, which is used to bind to the directory service.

ldapHostName
Specifies the (LDAP server host name. This host name is either an IP address or a domain name service (DNS) name.

ldapPort
Specifies a valid LDAP server port number.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask applyWizardSettings {-secureLocalResources true_or_false -secureApps true_or_false
-ignoreCase true_or_false -ldapServerType server_type -ldapBaseDN base_DN_value
-ldapBindDN bind_DN_value -ldapBindPassword bind_DN_password
-ldapHostName host_name -ldapPort port_number -userRegistryType
user_registry_type
-adminName administrator_user_name -adminPassword administrator_password}
```

- Using Jython string:

```
AdminTask.applyWizardSettings ('[-secureLocalResources true_or_false -secureApps true_or_false
-ignoreCase true_or_false -ldapServerType server_type -ldapBaseDN base_DN_value
-ldapBindDN bind_DN_value -ldapBindPassword bind_DN_password
-ldapHostName host_name -ldapPort port_number -userRegistryType
 user_registry_type
-adminName administrator_user_name -adminPassword administrator_password]')
```

- Using Jython list:

```
AdminTask.applyWizardSettings (['-secureLocalResources', 'true_or_false',
'-secureApps', 'true_or_false', '-ignoreCase', 'true_or_false',
'-ldapServerType', 'server_type', '-ldapBaseDN', 'base_DN_value',
'-ldapBindDN', 'bind_DN_value', '-ldapBindPassword', 'bind_DN_password',
'-ldapHostName', 'host_name', '-ldapPort', 'port_number',
'-userRegistryType', 'user_registry_type', '-adminName', 'administrator_user_name',
'-adminPassword', 'administrator_password'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask applyWizardSettings {-interactive}
```

- Using Jython string:

```
AdminTask.applyWizardSettings ('[-interactive]')
```

- Using Jython list:

```
AdminTask.applyWizardSettings (['-interactive'])
```

getCurrentWizardSettings

The **getCurrentWizardSettings** command retrieves the current security wizard settings from the workspace.

Parameters

None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getCurrentWizardSettings
```

- Using Jython string:

```
AdminTask.getCurrentWizardSettings
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getCurrentWizardSettings {-interactive}
```

- Using Jython string:

```
AdminTask.getCurrentWizardSettings ('[-interactive]')
```

isAdminLockedOut

The **isAdminLockedOut** command verifies that at least one administrative user exists in the input user registry.

Required parameters

registryType
Specifies a valid user registry type. The following type values are valid:

- `LDAPUserRegistry`

  This registry type uses the Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups exist in an external LDAP directory.
- `CustomUserRegistry`

  This type specifies a custom registry.
- `WIMUserRegistry`

  This value has the same effect as the Federated repositories option in the Security Configuration Wizard on the administrative console. This registry type manages identities in a single, virtual realm that is stored in multiple repositories.
- `LocalOSUserRegistry`

  This value specifies the registry for the local operating system of the application server.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask isAdminLockedOut {-registryType user_registry_type}
```

- Using Jython string:

```
AdminTask.isAdminLockedOut ('[-registryType user_registry_type]')
```

- Using Jython list:

```
AdminTask.isAdminLockedOut (['-registryType', 'user_registry_type'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask isAdminLockedOut {-interactive}
```

- Using Jython string:

```
AdminTask.isAdminLockedOut ('[-interactive]')
```

- Using Jython list:

```
AdminTask.isAdminLockedOut (['-interactive']
```

isAppSecurityEnabled

The **isAppSecurityEnabled** command returns a true or false value that indicates whether application security is enabled.

Parameters

None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask isAppSecurityEnabled
```

- Using Jython string:

```
AdminTask.isAppSecurityEnabled
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask isAppSecurityEnabled {-interactive}
```

- Using Jython string:

```
AdminTask.isAppSecurityEnabled ('[-interactive]')
```

isGlobalSecurityEnabled

The **isGlobalSecurityEnabled** command returns a true or false value that indicates whether administrative security is enabled.

Parameters

None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask isGlobalSecurityEnabled
```

- Using Jython string:

```
AdminTask.isGlobalSecurityEnabled
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask isGlobalSecurityEnabled {-interactive}
```

- Using Jython string:

```
AdminTask.isGlobalSecurityEnabled ('[-interactive]')
```

setGlobalSecurity

The **setGlobalSecurity** command changes whether administrative security is enabled.

Required parameters

enabled
Specifies whether to enable administrative security. This **enabled** parameter is equivalent to the Enable application security option on the administrative console.

You must specify either a true or false value.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setGlobalSecurity {-enabled true_or_false}
```

- Using Jython string:

```
AdminTask.setGlobalSecurity ('[-enabled true_or_false]')
```

- Using Jython list:

```
AdminTask.setGlobalSecurity (['-enabled', 'true_or_false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setGlobalSecurity {-interactive}
```

- Using Jython string:

```
AdminTask.setGlobalSecurity ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setGlobalSecurity (['-interactive'])
```

setUseRegistryServerId

The **setUseRegistryServerId** command updates the useRegistryServerId field in the user registry object within the `security.xml` file with a true or flase value. If you set the field value to true, the application server uses a user-specified server ID for interprocess communications.

Required parameters

useRegistryServerId
Specifies a true or false value for the useRegistryServerId setting.

useRegistryType
Specifies a valid user registry type. The following type values are valid:

- `LDAPUserRegistry`

  This registry type uses the Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups exist in an external LDAP directory.

- `CustomUserRegistry`

  This type specifies a custom registry.

- `WIMUserRegistry`

  This value has the same effect as the Federated repositories option in the Security Configuration Wizard on the administrative console. A registry type manages identities in a single, virtual realm that is stored in multiple repositories.
- `LocalOSUserRegistry`

  This value specifies the registry for the local operating system of the application server.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setUseRegistryServerId {-userRegistryType user_registry_type -useRegistryServerId
true_or_false}
```

- Using Jython string:

```
AdminTask.setUseRegistryServerId ('[-userRegistryType user_registry_type -useRegistryServerId
true_or_false]')
```

- Using Jython list:

```
AdminTask.setUseRegistryServerId (['-userRegistryType', 'user_registry_type', '-useRegistryServerId',
'true_or_false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setUseRegistryServerId {-interactive}
```

- Using Jython string:

```
AdminTask.setUseRegistryServerId ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setUseRegistryServerId (['-interactive'])
```

validateAdminName

The **validateAdminName** command verifies whether an administrator name exists in the input user registry.

Required parameters

adminUser
Specifies an administrative user name.

registryType
Specifies a valid user registry type. The following type values are valid:

- `LDAPUserRegistry`

  This registry type uses the Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups exist in an external LDAP directory.
- `CustomUserRegistry`

  This type specifies a custom registry.
- `WIMUserRegistry`

  This value has the same effect as the Federated repositories option in the Security Configuration Wizard on the administrative console. A registry type manages identities in a single, virtual realm that is stored in multiple repositories.
- `LocalOSUserRegistry`

  This value specifies the registry for the local operating system of the application server.

Optional parameters

ldapServerType
Specifies a valid LDAP server type. The following type values are valid:

- IBM_DIRECTORY_SERVER

  This value refers to a supported IBM Tivoli Directory Server version.
- IPLANET

  This value refers to a supported Sun Java System Directory Server version.
- NDS

  This value refers to a supported Novell eDirectory version.
- DOMINO502

  This value refers to a supported IBM Lotus Domino server version.
- SECUREWAY

  This value refers to an IBM SecureWay Directory Server version.
- ACTIVE_DIRECTORY

  This value refers to a supported Microsoft Active Directory version.
- CUSTOM

  This value refers to a custom registry implementation.

For more information about the supported LDAP server versions, see the WebSphere Application Server detailed system requirements documentation.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask validateAdminName {-ldapServerType server_type -registryType user_registry_type
-adminUser administrator}
```

- Using Jython string:

```
AdminTask.validateAdminName ('[-ldapServerType server_type -registryType user_registry_type
-adminUser administrator]')
```

- Using Jython list:

```
AdminTask.validateAdminName (['-ldapServerType', 'server_type', '-registryType',
'user_registry_type',
'-adminUser', 'administrator'])
```

Interactive mode example usage:

- Using Jacl:

  ```
  $AdminTask validateAdminName {-interactive}
  ```
- Using Jython string:

  ```
  AdminTask.validateAdminName ('[-interactive]')
  ```
- Using Jython list:

  ```
  AdminTask.validateAdminName (['-interactive'])
  ```

validateLDAPConnection

The **validateLDAPConnection** command validates the connection to a specified LDAP server.

Required parameters

hostname
Specifies the LDAP server host name. This host name is either an IP address or a domain name service (DNS) name.

sslEnabled
Specifies whether secure socket communications is enabled with the Lightweight Directory Access Protocol (LDAP) server. When this option is selected, LDAP Secure Sockets Layer (SSL) settings are used, if specified.

type
Specifies a valid LDAP registry type. The following type values are valid:

- `IBM_DIRECTORY_SERVER`

  This value refers to a supported IBM Tivoli Directory Server version.
- `IPLANET`

  This value refers to a supported Sun Java System Directory Server version.
- `NDS`

  This value refers to a supported Novell eDirectory version.
- `DOMINO502`

  This value refers to a supported IBM Lotus Domino server version.
- `SECUREWAY`

  This value refers to an IBM SecureWay Directory Server version.
- `ACTIVE_DIRECTORY`

  This value refers to a supported Microsoft Active Directory version.
- `CUSTOM`

  This value refers to a custom registry implementation.

For more information about the supported LDAP server versions, see the WebSphere Application Server detailed system requirements documentation.

Optional parameters

baseDN
Specifies the base distinguished name of the directory service, which indicates the starting point for LDAP searches in the directory service. For example, `ou=Rochester, o=IBM, c=us`

bindDN
Specifies the distinguished name for the application server, which is used to bind to the directory service.

bindPassword
Specifies the password for the application server, which is used to bind to the directory service.

port
Specifies the LDAP server port number.

securityDomainName
Specifies the name that is used to uniquely identify the security domain.

sslAlias
Specifies which SSL configuration to use for LDAP.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask validateLDAPConnection {-baseDN base_ND_value -bindDN bind_DN_value
-bindPassword bind_password -hostname host_name -securityDomainName
 security_domain_name
-port port_number -sslAlias alias -sslEnabled true_or_false
-type LDAP_registry_type}
```

- Using Jython string:

```
AdminTask.validateLDAPConnection ('[-baseDN base_ND_value -bindDN bind_DN_value
-bindPassword bind_password -hostname host_name -securityDomainName
security_domain_name
-port port_number -sslAlias alias -sslEnabled true_or_false
-type LDAP_registry_type]')
```

- Using Jython list:

```
AdminTask.validateLDAPConnection (['-baseDN', 'base_ND_value', '-bindDN', 'bind_DN_value',
'-bindPassword', 'bind_password', '-hostname', 'host_name', '-securityDomainName',
'security_domain_name', '-port', 'port_number', '-sslAlias', 'alias',
'-sslEnabled', 'true_or_false', '-type', 'LDAP_registry_type'])
```

Interactive mode example usage:

- Using Jacl:

  ```
  $AdminTask validateLDAPConnection {-interactive}
  ```

- Using Jython string:

  ```
  AdminTask.validateLDAPConnection ('[-interactive]')
  ```

- Using Jython list:

  ```
  AdminTask.validateLDAPConnection (['-interactive'])
  ```

WIMCheckPassword

The **WIMCheckPassword** command validates the user name and password in the federated repository.

Required parameters

username
Specifies the name of the user.

password
Specifies the password for the user.

Examples

Batch mode example usage:

- Using Jacl:

  ```
  $AdminTask.WIMCheckPassword {-username user_name -password password}
  ```

- Using Jython string:

  ```
  AdminTask.WIMCheckPassword ('[-username user_name -password password]')
  ```

- Using Jython list:

  ```
  AdminTask.WIMCheckPassword (['-username', 'user_name', '-password', 'password'])
  ```

Interactive mode example usage:

- Using Jacl:

  ```
  $AdminTask WIMCheckPassword {-interactive}
  ```

- Using Jython string:

  ```
  AdminTask.WIMCheckPassword ('[-interactive]')
  ```

- Using Jython list:

  ```
  AdminTask.WIMCheckPassword (['-interactive'])
  ```

Configuring multiple security domains using scripting

You can customize your security configuration at the cell, sever, or cluster level by configuring multiple security domains.

Before you begin

Users assigned to the administrator role can configure security domains. Verify that you have the appropriate administrative role before configuring security domains. Also, enable global security in your environment before configuring multiple security domains.

About this task

You can create multiple security domains to customize your security configuration. Use multiple security domains to achieve the following goals:

- Configure different security attributes for administrative and user applications within a cell
- Consolidate server configurations by managing different security configurations within a cell
- Restrict access between applications with different user registries, or configure trust relationships between applications to support communication across registries

Procedure

1. Create a security domain. Create multiple security domains in your configuration. By creating multiple security domains, you can configure different security attributes for administrative and user applications within a cell environment.
2. Assign the security domain to one or a set of resources or scopes. Assign management resources to security domains. Set management resources to your security domains to customize your security configuration for a cell, server, or cluster.
3. Customize your security configuration by specifying attributes for your security domain. See the following examples of security attributes:
   - User registries to validate user credentials
   - Authorization for validating access to resources
   - Trust association interceptor (TAI) to authenticate a web user using a reverse proxy server
   - Application and system JAAS login configurations
   - LTPA timeout settings
   - Application security enablement to provide application isolation and requirements for authenticating application users
   - Java 2 Security to increase overall system integrity by checking for permissions before allowing access to certain protected system resources
   - Remote Method Invocation over Internet Inter-ORB Protocol (RMI/IIOP) to invoke web services through remote procedure calls
   - Custom properties

Configuring security domains using scripting

Use this topic to create multiple security domains in your configuration. By creating multiple security domains, you can configure different security attributes for administrative and user applications within a cell environment.

Before you begin

You must have the administrator role to configure security domains. Also, enable global security in your environment before configuring multiple security domains.

About this task

You can create multiple security domains to customize your security configuration. Use multiple security domains to achieve the following goals:

- Configure different security attributes for administrative and user applications within a cell

- Consolidate server configurations by managing different security configurations within a cell
- Restrict access between applications with different user registries, or configure trust relationships between applications to support communication across registries

Use the following steps to create a new security domain with the wsadmin tool:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Create a security domain.

   To create a security domain, you can create a new security domain, copy an existing security domain, or copy the existing global security configuration.

   - Use the createSecurityDomain command to create the domain-security.xml and domain-security-map.xml security files in the *profile_root*/config/waspolicies/default/*security_configuration_name* directory. No configuration data is added to the domain-security.xml file. Use the following Jython command example to create a security domain:

```
AdminTask.createSecurityDomain('-securityDomainName securityDomain1 -securityDomainDescription "handles user applications"')
```

     The command returns the object name of the security domain that was created, as the following example output demonstrates:

```
'waspolicies/default/securitydomains/mydomain:domain-security.xml#AppSecurity_1183132319126'
```

   - Use the copySecurityDomain command to create a new security domain with the attributes of an existing security domain. If the security configuration of the existing domain has an active user registry defined, then a new realm name for that registry must be used in the new security configuration. If a realm name is not specified with the copySecurityDomain command, then the command assigns a name.

Table 1. copySecurityDomain command parameter descriptions. Specify the following parameters to copy an existing security domain:

Parameter	Description
-securityDomainName	Specifies the name of the new security domain to create (String, required)
-copyFromSecurityDomainName	Specifies the name of the existing security domain to copy (String, required)
-realmName	Specifies the name of the realm in the security domain to create. The system assigns the realm name to the active user registry in the security domain (String, optional)
-securityDomainDescription	Specifies a description for the security domain to create (String, optional)

     Use the following Jython command to copy an existing security domain:

```
AdminTask.copySecurityDomain('-securityDomainName copyOfDomain1 -copyFromSecurityDomainName securityDomain1')
```

     The command returns the object name of the new security domain, as the following example output demonstrates:

```
'waspolicies/default/securitydomains/copyOfDomain1:domain-security.xml#AppSecurity_1183132319186'
```

   - Use the copySecurityDomainFromGlobalSecurity command to create a security domain by copying the global security configuration. If the global security configuration has an active user registry defined, then a new realm name for that registry must be used for the security domain to create. If you do not specify a realm name, then the command assigns a name.

Table 2. copySecurityDomainFromGlobalSecurity command parameter descriptions. Specify the following parameters to copy the global security configuration:

Parameter	Description
-securityDomainName	Specifies the name of the new security domain to create (String, required)
-realmName	Specifies the name of the realm in the security domain to create. The system assigns the realm name to the active user registry in the security domain (String, optional)
-securityDomainDescription	Specifies a description for the security domain to create (String, optional)

     Use the following Jython command to copy the global security configuration:

```
AdminTask.copySecurityDomainFromGlobalSecurity('-securityDomainName GScopy')
```

The command returns the object name of the new security domain, as the following example output demonstrates:

```
'waspolicies/default/securitydomains/copyOfDomain1:domain-security.xml#AppSecurity_1183132319186'
```

3. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

What to do next

Use the wsadmin tool to map a scope to your security domain. Additionally, you can configure security artifacts in the newly created domain, by:

- configuring user registries.
- enabling application and Java EE security.
- setting Lightweight Third-Party Authentication (LTPA) timeout.
- configuring System and Application Java™ Authentication and Authorization Service (JAAS) login.
- configuring Java 2 Connector (J2C) authorization data.
- configuring Remote Method Invocation over Internet Inter-ORB Protocol (RMI/IIOP) security.

Configuring local operating system user registries using scripting

Use this topic to configure user registries for global security and security domain configurations using the wsadmin tool. You can define user registries at the global level and for multiple security domains.

Before you begin

You must meet the following requirements before configuring local operating system user registries:

- You must have the administrator or new admin role.
- Enable global security in your environment.
- To configure local operating system user registries for multiple security domains, you must configure at least one security domain.

About this task

Configure local operating system user registries to support use of the authentication mechanism with the user accounts database of the local operating system. You can specify local operating system user registries at the global level and at the security domain.

When you configure a user registry in the global security configuration, the administrator does not specify a realm name for the user registry. The system determines the realm name from the security runtime. The system typically specifies the hostname for local operating system registries.

In security domains, you can configure a different realm for a user registry configuration. For example, you can configure two registries that use the same LDAP server listening on the same port, but use different base distinguished names (baseDN). This allows the configuration to serve different sets of users and groups. To use this type of scenario, you must specify a realm name for each user registry configured for a domain. Because there can be multiple realms in your configuration, you can also specify a list of trusted realms. This allows communication between applications that use different realms.

Use the following steps to configure local operating system user registries for your global security configuration and for multiple security domains:

Procedure

- Configure local operating system registries for global security configurations.

1. Use the configureAdminLocalOSUserRegistry command and the following optional parameters to configure a local operating system user registry in your global security configuration.

   Note: This command is not supported in a local mode.

Table 3. Optional parameters. This table lists the configureAdminLocalOSUserRegistry command and its optional parameters:

Parameter	Description	Data type
-autoGenerateServerId	Specifies whether to automatically generate the server identity to use for internal process communication. To set a specific server identity, specify the -serverId parameter.	Boolean
-serverId	Specifies the user identity in the repository to use for internal process communication.	String
-serverIdPassword	Specifies the password that corresponds to the user identity.	String
-primaryAdminId	Specifies the name of the user with administrative privileges as defined in the registry. This parameter does not apply to security configurations. The user name must exist in the user registry repository.	String
-customProperties	Specifies a list of attribute and value pairs to store as custom properties on the user registry. Separate each attribute and value pair with a comma character (,), as the following syntax displays: `"attribute1=value1","attribute2=value2"`	String
-verifyRegistry	Specifies whether to verify the user registry. The default value is `true` and verification is automatically performed.	Boolean
-ignoreCase	Specifies whether to perform the case-sensitive authorization check. This only applies to the z/OS® local operating system user registry.	Boolean

   Use the following Jython example command to configure the local operating system registry for global security:

```
AdminTask.configureAdminLocalOSUserRegistry('-autoGenerateServerId true -primaryAdminId gsAdmin')
```

2. Configure the user registry to be the active user registry for the server.

   For example, the following Jython command sets the active user registry as the `LocalOSUserRegistry` registry for your global security configuration:

```
AdminTask.setAdminActiveSecuritySettings('-activeUserRegistry LocalOSUserRegistry')
```

3. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

- Configure local operating system registries for security domains.
  1. Determine the name of the security domain to configure.

     Use the listSecurityDomains command to list all security domains on the server, as the following Jython example demonstrates:

```
AdminTask.listSecurityDomains()
```

     If you want to configure the local operating system registry for a specific server, cluster, or cell, use the getSecurityDomainForResource command to display the security domain name for the management scope of interest. The following Jython example displays the name of the security domain configured at the cell-level:

```
AdminTask.getSecurityDomainForResource('-resourceName Cell=:Node=myNode:Server=myServer')
```

     For this example, the command returns the following output:

```
domain2
```

2. Configure a local operating system user registry for a security domain. Use the configureAppLocalOSUserRegistry command and the following optional parameters to configure a local operating system user registry.

   Note: This command is not supported in a local mode.

Table 4. Optional parameters. This table describes the configureAppLocalOSUserRegistry command and its optional parameters:

Parameter	Description	Data type
-securityDomainName	Specifies the unique name that identifies the security domain of interest.	String
-realmName	Specifies the name of the realm of the user registry.	String
-customProperties	Specifies a list of attribute and value pairs to store as custom properties on the user registry object. Separate each attribute and value pair with a comma character (,).	String
-verifyRegistry	Specifies whether to verify the user registry. The default value is `true`, and verification is automatically performed.	Boolean
-ignoreCase	Specifies whether to perform the case-sensitive authorization check. This only applies to the z/OS local operating system user registry.	Boolean

   Use the following Jython command to configure the local operating system user registry for the `domain2` security domain:

```
AdminTask.configureAppLocalOSUserRegistry('-securityDomainName domain2 -realmName domain2Realm')
```

3. Configure the user registry to be the active user registry for the server.

   For example, the following Jython command sets the active user registry as the `LocalOSUserRegistry` registry for your security domain configuration:

```
AdminTask.setAppActiveSecuritySettings('-securityDomainName domain2 -activeUserRegistry LocalOSUserRegistry')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

What to do next

Configuring custom user registries using scripting

Use this topic to configure custom user registries for global security and security domain configurations using the wsadmin tool. You can define custom user registries at the global level and for multiple security domains.

Before you begin

You must meet the following requirements before configuring custom user registries:

- You must have the administrator or new admin role.
- Enable global security in your environment.
- Implement and build the UserRegistry interface and configure a custom registry.
- To configure custom user registries for multiple security domains, you must configure at least one security domain.

About this task

WebSphere Application Server security supports stand-alone custom registries in addition to the local operating system registry, standalone Lightweight Directory Access Protocol (LDAP) registries, and federated repositories for authentication and authorization. A stand-alone custom-implemented registry uses the UserRegistry Java interface as provided by the product. A stand-alone custom registry can

support any type of account repository from a relational database, flat file, and so on. You can specify custom user registries at the global level and at the security domain.

When you configure a user registry in the global security configuration, the administrator does not specify a realm name for the user registry. The system determines the realm name from the security run time. The realm name for custom registries is set by the custom registry.

Use the following command to make a specific user registry the active user registry in the global security configuration:

Jython

```
AdminTask.setAdminActiveSecuritySettings ('[-activeUserRegistry CustomUserRegistry]')
```

Jacl

```
$AdminTask setAdminActiveSecuritySettings {-activeUserRegistry CustomUserRegistry}
```

Use the following command to make a specific user registry the active user registry in the security domain configuration:

Jython

```
AdminTask.setAppActiveSecuritySettings ('[-securityDomainName domain2 -activeUserRegistry CustomUserRegistry]')
```

Jacl

```
$AdminTask setAppActiveSecuritySettings {-securityDomainName domain2 -activeUserRegistry CustomUserRegistry}
```

In security domains, you can configure a different realm for a user registry configuration. For example, you can configure two registries that use the same LDAP server listening on the same port, but use different base distinguished names (baseDN). This method supports the configuration to serve different sets of users and groups. To use this type of scenario, you must specify a realm name for each user registry configured for a domain. Multiple realms can exist in your configuration, and you can also specify a list of trusted realms. Communications between applications that use different realms is supported.

Use the following steps to configure custom user registries for your global security configuration and for multiple security domains:

Procedure

- Configure custom user registries for global security configurations.

  Note: This command is not supported in a local mode.

Table 5. Optional parameters. Use the configureAdminCustomUserRegistry command and the following optional parameters to configure a custom user registry in your global security configuration:

Parameter	Description	Data Type
-autoGenerateServerId	Specifies whether to automatically generate the server identity to use for internal process communication. To set a specific server identity, specify the -serverId parameter.	Boolean
-serverId	Specifies the user identity in the repository to use for internal process communication.	String
-serverIdPassword	Specifies the password that corresponds to the user identity.	String
-primaryAdminId	Specifies the name of the user with administrative privileges as defined in the registry. This parameter does not apply to security configurations. The user name must exist in the user registry repository.	String
-customRegClass	Specifies the class name that implements the UserRegistry interface in the com.ibm.websphere.security class.	String
-ignoreCase	Specifies whether to require case sensitive authorization. Specify `true` to ignore case during authorization.	Boolean

Table 5. Optional parameters (continued). Use the configureAdminCustomUserRegistry command and the following optional parameters to configure a custom user registry in your global security configuration:

Parameter	Description	Data Type

-customProperties	Specifies a list of attribute and value pairs to store as custom properties on the user registry object. Separate each attribute and value pair with a comma character. Also, separately surround the attribute and value pairs with bracket characters ([]) for the Jython programming language and brace characters ({}) for the Jacl programming language. For example:
	Jython `-customProperties ["attribute1=value1",`
	`"attribute2=value2"]`
	Jython `-customProperties {"attribute1=value1",`
	`"attribute2=value2"}`
		String

-verifyRegistry	Specifies whether to verify the user registry. The default value is `true` and verification is automatically performed.	Boolean

Use the following example command to configure the custom user registry for global security:

Jython

```
AdminTask.configureAdminCustomUserRegistry ('[-autoGenerateServerId true -primaryAdminId gsAdmin
 -customProperties ["attribute1=value1","attribute2=value2"]]')
```

Jacl

```
$AdminTask configureAdminCustomUserRegistry {-autoGenerateServerId true -primaryAdminId gsAdmin
 -customProperties {"attribute1=value1","attribute2=value2"}}
```

- Configure custom user registries for security domains.
  1. Determine the name of the security domain to configure.

     Use the listSecurityDomains command to list all security domains on the server:

     ### Jython

     ```
     AdminTask.listSecurityDomains()
     ```

     ### Jacl

     ```
     $AdminTask listSecurityDomains
     ```

  2. Configure a custom user registry for a security domain.

     Note: This command is not supported in a local mode.

Table 6. Optional parameters. Use the configureAppCustomUserRegistry command and the following optional parameters to configure a custom user registry:

Parameter	Description	Data type
-securityDomainName	Specifies the unique name that identifies the security domain of interest.	String
-realmName	Specifies the name of the realm of the user registry.	String
-customRegClass	Specifies the class name that implements the UserRegistry interface in the com.ibm.websphere.security class.	String
-ignoreCase	Specifies whether to require case sensitive authorization. Specify `true` to ignore case during authorization.	Boolean

-customProperties	Specifies a list of attribute and value pairs to store as custom properties on the user registry object. Separate each attribute and value pair with a comma character. Also, separately surround the attribute and value pairs with bracket characters ([]) for the Jython programming language and brace characters ({}) for the Jacl programming language. For example:
	Jython `-customProperties ["attribute1=value1",`
	`"attribute2=value2"]`
	Jython `-customProperties {"attribute1=value1",`
	`"attribute2=value2"}`
		String

-verifyRegistry	Specifies whether to verify the user registry. The default value is `true` and verification is automatically performed.	Boolean

Use the following example command to configure the custom user registry for the domain2 security domain:

Jython

```
AdminTask.configureAppCustomUserRegistry ('[-securityDomainName domain2 -realmName domain2Realm
 -customProperties ["attribute1=value1","attribute2=value2"]]')
```

Jacl

```
$AdminTask configureAppCustomUserRegistry {-securityDomainName domain2 -realmName domain2Realm
 -customProperties {"attribute1=value1","attribute2=value2"}}
```

What to do next

Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Configuring JAAS login modules using wsadmin scripting

Use this topic to use the wsadmin tool to configure and manage Java Authentication and Authorization Service (JAAS) login entries to allow communication between realms in a multiple security domain environment.

Before you begin

You must meet the following requirements before configuring local operating system user registries:

- You must have the administrator or new admin role.
- Enable global security in your environment.
- Configure multiple realms using security domains in your environment.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client topic for more information.
2. Configure a JAAS login module.

   Use the configureJAASLoginEntry command to configure a Java Authentication and Authorization Service (JAAS) login entry in a security domain or in the global security configuration. You can use this command to modify existing JAAS login entries or to create new login entries.

   Specify the following parameters to configure the JAAS login module:

Table 7. Command parameters. Run the configureJAASLoginEntry command to configure a JAAS login module.

Parameter	Description
-loginEntryAlias	Specifies an alias that identifies the JAAS login entry in the configuration. (String, required)
-loginType	Specifies the type of JAAS login entry of interest. Specify system for the system login type or application for the application login type. (String, required)
-securityDomainName	Specifies the name of the security configuration. If you do not specify a security domain name, the system updates the global security configuration. (String, optional)
-loginModules	Specifies a comma (,) separated list of login module class names. Specify the list in the order that the system calls them. (String, optional)

Table 7. Command parameters (continued). Run the configureJAASLoginEntry command to configure a JAAS login module.

Parameter	Description
-authStrategies	Optionally specifies the authentication behavior as authentication proceeds down the list of login modules. (String, optional) Specify one or many of the following values in a comma (,) separated list: • REQUIRED Specifies that the LoginModule module is required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list for each realm. • REQUISITE Specifies that the LoginModule module is required to succeed. If authentication is successful, the process continues down the LoginModule list in the realm entry. If authentication fails, control immediately returns to the application. Authentication does not proceed down the LoginModule list. • SUFFICIENT Specifies that the LoginModule module is not required to succeed. If authentication succeeds, control immediately returns to the application. Authentication does not proceed down the LoginModule list. If authentication fails, the process continues down the list. • OPTIONAL Specifies that the LoginModule module is not required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list.

Use the configureJAASLoginEntry command to configure the JAAS login module, as the following Jython example demonstrates:

```
AdminTask.configureJAASLoginEntry('[-securityDomainName testDomain
-loginType application -loginEntryAlias testLoginEntry -loginModules
"com.ibm.ws.security.common.auth.module.WSLoginModuleImpl" -authStrategies "REQUIRED"]')
```

3. Set custom properties for the JAAS login module.

   Use the configureLoginModule command to specify custom properties, modify the authentication strategy, or set the module to use a login module proxy. The following Jython command sets the `debug` and `delegate` custom properties for the `testLoginEntry` JAAS login entry:

```
AdminTask.configureLoginModule('[-securityDomainName testDomain -loginType application
-loginEntryAlias testLoginEntry -loginModule com.ibm.ws.security.common.auth.module.WSLoginModuleImpl
-customProperties ["debug=true","delegate=WSLogin"]]')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Configuring Common Secure Interoperability authentication using scripting

Use this topic to use the wsadmin tool to configure inbound and outbound communications using the Common Secure Interoperability protocol. Common Secure Interoperability Version 2 (CSIv2) supports increased vendor interoperability and additional features.

Before you begin

You must meet the following requirements before configuring local operating system user registries:

- You must have the administrator or new admin role.
- Enable global security in your environment.
- Configure multiple realms using security domains in your environment.

Procedure

- Configure CSI inbound communication authentication.

  Inbound authentication refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the interoperable object reference (IOR) that the client retrieves from the name server.

  1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.

2. Determine the settings to specify for CSI inbound communication.

   The configureCSIInbound command configures various settings for CSI inbound communication.

Table 8. Command parameters. Review the following list of optional parameters to determine the attributes to set in your configuration:

Parameter	Description
-securityDomainName	Specifies the name of the security configuration. If you do not specify a security domain name, the command modifies the global security configuration. (String)
-messageLevelAuth	Specifies whether clients connecting to this server must specify a user ID and password. Specify `Never` to disable the user ID and password requirement. Specify `Supported` to accept a user ID and password. Specify `Required` to require a user ID and password. (String)
-supportedAuthMechList	Specifies the authentication mechanism to use. Specify `KRB5` for Kerberos authentication, `LTPA` for Lightweight Third-Party Authentication, `BasicAuth` for basic authentication, and `custom` to use your own authentication token implementation. You can specify more than one, separated by the pipe character (`|`). (String)
-clientCertAuth	Specifies whether a client that connects to the server must connect using an SSL certificate. Specify `Never` to allow clients to connect without SSL certificates. Specify `Supported` to accept clients connecting with and without SSL certificates. Specify `Required` to require clients to use SSL certificate. (String)
-transportLayer	Specifies the transport layer support level. Specify `Never` to disable transport layer support. Specify `Supported` to enable transport layer support. Specify `Required` to require transport layer support. (String)
-sslConfiguration	Specifies the SSL configuration alias to use for inbound transport. (String)
-enableIdentityAssertion	Specifies whether to enable identity assertion. When using the identity assertion authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element. Specify `true` for the **-enableIdentityAssertion** parameter to enable identity assertion. (Boolean)
-trustedIdentities	Specifies a list of trusted server identities, separated by the pipe character (`|`). To specify a null value, set the value of the **-trustedIdentities** parameter as an empty string (`""`). (String)
-statefulSession	Specifies whether to enable a stateful session. Specify `true` to enable a stateful session. (Boolean)
-enableAttributePropagation	Specifies whether to enable security attribute propagation. Security attribute propagation allows the application server to transport authenticated subject contents and security context information from one server to another in your configuration. Specify `true` to enable security attribute propagation. (Boolean)

3. Configure CSI inbound communication authentication.

   The configureCSIInbound command configures the CSIv2 Inbound authentication on a security domain or on the global security configuration. When configuring CSI Inbound in a security domain for the first time, the CSI objects are copied from global security. Then, the changes are applied to configuration.

   Use the configureCSIInbound command to configure CSI inbound authentication for a security domain or the global security configuration, as the following Jython example demonstrates:

```
AdminTask.configureCSIInbound('-securityDomainName testDomain -messageLevelAuth Supported
-supportedAuthMechList KRB5|LTPA -clientCertAuth Supported -statefulSession true')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

- Configure CSI outbound communication authentication.

  Outbound authentication refers to the configuration that determines the type of authentication that is performed for outbound requests to downstream servers.

  1. Start the wsadmin scripting tool.
  2. Determine the settings to specify for CSI outbound communication.

     The configureCSIOutbound command configures various settings for CSI outbound communication.

Table 9. Command parameters. Review the following list of optional parameters to determine the attributes to set in your configuration:

Parameter	Description
-securityDomainName	Specifies the name of the security configuration. If you do not specify a security domain name, the command modifies the global security configuration. (String)
-enableAttributePropagation	Specifies whether to enable security attribute propagation. Security attribute propagation allows the application server to transport authenticated subject contents and security context information from one server to another in your configuration. Specify `true` to enable security attribute propagation. (Boolean)

Table 9. Command parameters (continued). Review the following list of optional parameters to determine the attributes to set in your configuration:

Parameter	Description
-enableIdentityAssertion	Specifies whether to enable identity assertion. When using the identity assertion authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element. Specify `true` for the **-enableIdentityAssertion** parameter to enable identity assertion. (Boolean)
-useServerIdentity	Specifies whether to use the server identity to establish trust with the target server. Specify `true` to use the server identity. (Boolean)
-trustedId	Specifies the trusted identity that the application server uses to establish trust with the target server. (String)
-trustedIdentityPassword	Specifies the password of the trusted server identity. (String)
-messageLevelAuth	Specifies whether clients connecting to this server must specify a user ID and password. Specify `Never` to disable the user ID and password requirement. Specify `Supported` to accept a user ID and password. Specify `Required` to require a user ID and password. (String)
-supportedAuthMechList	Specifies the authentication mechanism to use. Specify `KRB5` for Kerberos authentication, `LTPA` for Lightweight Third-Party Authentication, `BasicAuth` for basic authentication, and `custom` to use your own authentication token implementation. You can specify more than one, separated by the pipe character (|). (String)
-clientCertAuth	Specifies whether a client that connects to the server must connect using an SSL certificate. Specify `Never` to allow clients to connect without SSL certificates. Specify `Supported` to accept clients connecting with and without SSL certificates. Specify `Required` to require clients to use SSL certificate. (String)
-transportLayer	Specifies the transport layer support level. Specify `Never` to disable transport layer support. Specify `Supported` to enable transport layer support. Specify `Required` to require transport layer support. (String)
-sslConfiguration	Specifies the SSL configuration alias to use for inbound transport. (String)
-statefulSession	Specifies whether to enable a stateful session. Specify `true` to enable a stateful session. (Boolean)
-enableCacheLimit	Specifies whether to limit the size of the CSIv2 session cache. If you specify a `true` value, a limit is added to the cache size. The value of the limit is determined by the values that you set with the **-maxCacheSize** and **-idleSessionTimeout** parameters. A false value, which is the default, does not limit the cache size. Consider adding a true value for this parameter if your environment uses Kerberos authentication and the clock skew for the configured key distribution center (KDC) is small. A small clock skew is defined as less than 20 minutes. This parameter applies when you set the **-statefulSession** parameter to true. (Boolean)
-maxCacheSize	Specifies the maximum size of the session cache after which expired sessions are deleted from the cache. Expired sessions are sessions that are idle longer than the time that you specify for the **-idleSessionTimeout** parameter. Consider specifying a value for this parameter if your environment uses Kerberos authentication and the clock skew for the configured key distribution center (KDC) is small. A small clock skew is defined as less than 20 minutes. Consider increasing the value of this parameter if the small cache size causes the garbage collection to run so frequently that it impacts the performance of the application server. This parameter applies when you set the **-statefulSession** and **-enableCacheLimit** parameters to true and set a value for the **-idleSessionTimeout** parameter. The valid range of values for this parameter is 100 to 1000. (Integer)
-idleSessionTimeout	Specifies the time, in milliseconds, that a CSIv2 session can remain idle before being deleted. The session is deleted if you set the **-enableCacheLimit** parameter to true and the value of the **-maxCacheSize** parameter is exceeded. Consider decreasing the value for this parameter if your environment uses Kerberos authentication and the clock skew for the KDC is small. A small clock skew can result in a greater number of rejected CSIv2 sessions. However, with a smaller value for this parameter, the application server can clean out the rejected sessions more often and reduce the possibility of a resource shortage. The valid range of values for this parameter is 60,000 to 86,400,000 milliseconds. (Integer)
-enableOutboundMapping	Specifies whether to enable custom outbound identity mapping. Specify `true` to enable custom outbound identity mapping. (Boolean)
-trustedTargetRealms	Specifies a list of target realms to trust. Separate each realm name with the pipe character (|). (String)

3. Configure CSI outbound communication authentication.

   The configureCSIOutbound command configures the CSIv2 outbound authentication in a security domain or in the global security configuration. When configuring CSI outbound authentication in a security domain for the first time, the application server copies the CSI objects from global security. Then, the application server applies the changes to that configuration.

   Use the configureCSIOutbound command to configure CSI outbound authentication for a security domain or the global security configuration, as the following Jython example demonstrates:

```
AdminTask.configureCSIOutbound('-securityDomainName testDomain -enableIdentityAssertion true
-trustedId myID -trustedIdentityPassword myPassword123 -messageLevelAuth Required
-trustedTargetRealms realm1|realm2|realm3')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Configuring trust association using scripting

Use the wsadmin tool to configure and manage trust association configurations in a multiple security domain environment. Trust association enables the integration of the application server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Before you begin

You must meet the following requirements before configuring a trust association:

- You must have the administrator or new admin role.
- Enable global security in your environment.
- Configure multiple realms using security domains in your environment.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure a trust association.

   Use the configureTrustAssociation command to enable the trust association. You can also use this command to create or modify a trust association interceptor.

   The following Jython command creates a trust association for the `testDomain` security domain and configures the trust association to act as a reverse proxy server:

```
AdminTask.configureTrustAssociation('-securityDomainName testDomain -enable true')
```

3. Configure the trust association interceptor.

   Use the configureInterceptor command to modify an existing interceptor. The following Jython command uses a WebSEAL interceptor to configure single sign-on for the `testDomain` security domain:

```
AdminTask.configureInterceptor('[-interceptor com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus
-securityDomainName testDomain -customProperties
["com.ibm.websphere.security.trustassociation.types=webseal",
"com.ibm.websphere.security.webseal.loginId=websealLoginID",
"com.ibm.websphere.security.webseal.id=iv-user"]]')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Mapping resources to security domains using scripting

Use this topic to assign management resources to security domains. Set management resources to your security domains to customize your security configuration for a cell, server, or cluster.

Before you begin

Users assigned to the administrator role can configure security domains. Verify that you have the appropriate administrative role before configuring security domains. Also, create a security domain, or copy an existing security domain before assigning resources to a security domain.

About this task

After creating a security domain, you can map management resources to the security domain. You can assign resources to a security domain at the server, cell, and cluster level. Use the following steps to assign a resource to a security domain:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Determine which security domain to map a resource.

   Use the listSecurityDomains command to view a list of security domains in your configuration. Specify `true` for the optional -listDescription parameter to list the description for each security domain, as the following Jython example demonstrates:

```
print AdminTask.listSecurityDomains('-listDescription true')
```

   The command returns the following example attribute list output:

```
{{name myDomain}
{description {security domain for administrative applications}}}
{{name domain2}
{description {new domain for cell123}}}
```

3. Assign a resource to a security domain.

   Use the mapResourceToSecurityDomain command to assign a management resource to the security domain. For example, use the following Jython command to secure all applications on the `server1` cell with the security attributes in the `domain2` security domain:

```
AdminTask.mapResourceToSecurityDomain('-securityDomainName domain2 -resourceName Cell=myCell:Node=myNode:Server=server1')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Results

Your security domain is updated in your configuration. All applications in the specified resource use the security attributes specified by the security domain. If the security domain does not contain all security attributes, then the missing attributes are obtained from the global security configuration.

What to do next

Restart each resource that you assigned to a security domain.

Removing resources from security domains using scripting

Use this topic to remove management resources from security domains. Remove all resources from a security domain before deleting the security domain from your configuration.

Before you begin

Users assigned to the administrator role can configure security domains. Verify that you have the appropriate administrative role before configuring security domains.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Determine the security domain to edit.

   Use the listSecurityDomains command to view a list of security domains in your configuration. Specify `true` for the optional -listDescription parameter to list the description for each security domain, as the following Jython example demonstrates:

```
print AdminTask.listSecurityDomains('-listDescription true')
```

   The command returns the following example output:

```
myDomain - security domain for administrative applications
domain2 - new domain for cell123
```

3. Verify the management resources that are assigned to the security domain.

Use the listResourcesInSecurityDomain command to view a list of resources that are mapped to the security domain of interest, as the following Jython example demonstrates:

```
print AdminTask.listResourcesInSecurityDomain('-securityDomainName domain2')
```

The command returns the following example output:

```
"Cell=myhostCell01"
```

4. Remove the resource from the security domain.

   Use the removeResourceFromSecurityDomain command to remove a management resource from the security domain. For example, use the following Jython command to remove the `Cell01` cell resource from the `domain2` security domain:

```
AdminTask.removeResourceFromSecurityDomain('-securityDomainName domain2 -resourceName Cell=myhostCell01')
```

5. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

What to do next

Restart each management resource that you removed from a security domain.

Removing security domains using scripting

Use this topic to delete security domains from your configuration using the wsadmin tool. Remove security domains that are not needed in your security configuration.

Before you begin

Users assigned to the administrator role can configure security domains. Verify that you have the appropriate administrative role before configuring security domains. A security domain must exist in your configuration.

Procedure

1. Start the wsadmin scripting tool.
2. Determine the security domain to delete.

   Use the listSecurityDomains to view a list of security domains in your configuration. Specify `true` for the optional -listDescription parameter to list the description for each security domain, as the following Jython example demonstrates:

```
print AdminTask.listSecurityDomains('-listDescription true')
```

   The command returns the following example output:

```
{{name myDomain}
{description {security domain for administrative applications}}}
{{name domain2}
{description {new domain for cell123}}}
```

3. Verify that no resources are assigned to the security domain to delete.

   You can use this step to manually remove resources from the security domain of interest. You do not need to complete this step if you want to delete the security domain and each assigned resource. Use the listResourcesInSecurityDomain command to view a list of resources that are mapped to the security domain of interest, as the following Jython example demonstrates:

```
print AdminTask.listResourcesInSecurityDomain('-securityDomainName domain2')
```

   If the command returns the name of a resource, use the removeResourceFromSecurityDomain command to remove a resource from the security domain. For example, use the following Jython command to remove the `Cell01` cell resource from the `domain2` security domain:

```
"AdminTask.removeResourceFromSecurityDomain('-securityDomainName domain2 -resourceName Cell=myhostCell01')"
```

4. Delete the security domain from your configuration.

   Use the deleteSecurityDomain command to delete the security domain. If a resource associated with the domain was deleted from the system but the mapping was not removed form the domain, specify the optional -force parameter to remove the domain, as the following Jython example demonstrates:

```
AdminTask.deleteSecurityDomain('-securityDomainName domain2 -force true')
```

5. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

Removing user registries using scripting

You can use the wsadmin tool to remove user registries from global security or security domain configurations. Use the steps in this topic to remove Lightweight Directory Access Protocol (LDAP), local operating system, custom, or federated repository user registries from your global security or security domain configurations.

Before you begin

You must meet the following requirements before configuring local operating system user registries:

- You must have the administrator or new admin role.
- Enable global security in your environment.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Determine the registry to remove.

   Use the getUserRegistryInfo command to display information about a user registry from the global security configuration or in a security domain. You must specify the type of user registry of interest. Valid values are `LDAPUserRegistry`, `WIMUserRegistry`, `CustomUserRegistry`, and `LocalOSUserRegistry`. The following command returns a list of values in the local operating system user registry object for the `domain2` security domain, as the following example Jython demonstrates:

```
AdminTask.getUserRegistryInfo('-securityDomainName domain2 -userRegistryType LocalOSUserRegistry')
```

3. Determine whether the registry of interest is the active user registry.

   You cannot remove the active user registry. Use the getActiveSecuritySettings command to see check if the user registry is the active user registry before removing it.

4. Remove the registry of interest.

   Use the unconfigureUserRegistry command to remove the registry of interest. If you remove the user registry from the global security configuration, then the command reduces the registry object to the minimum values for the configuration. If you remove the user registry from a security domain, then the command removes the configuration object from the security domain. The following Jython example removes the local operating system user registry configuration from the `domain2` security domain:

```
AdminTask.unconfigureUserRegistry('-securityDomainName domain2 -userRegistryType LocalOSUserRegistry')
```

5. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

SecurityDomainCommands command group for the AdminTask object

You can use the Jython scripting language to configure and administer security domains with the wsadmin tool. Use the commands and parameters in the SecurityDomainCommands group to create and manage security domains, assign servers and clusters to security domains as resources, and to query the security domain configuration.

Use the following commands to administer the security domain configuration:

- “convertServerSecurityToSecurityDomain” on page 110
- copySecurityDomain
- copySecurityDomainFromGlobalSecurity
- createSecurityDomain

- deleteSecurityDomain
- getSecurityDomainForResource
- listResourcesInSecurityDomain
- listSecurityDomains
- “listSecurityDomainsForResources” on page 115
- mapResourceToSecurityDomain
- modifySecurityDomain
- removeResourceFromSecurityDomain

convertServerSecurityToSecurityDomain

Starting in WebSphere Application Server Version 7.0, the use of security domains can be used in place of server level security configurations. If a server level security configuration is currently being used then the convertServerSecurityToSecurityDomain command can be used to covert it to a security domain.

The command creates a security domain and adds any security settings that are specified in the server level security configuration to the newly-created security domain. The server resources are mapped to the security domain.

Target object

None.

Required parameters

-serverResource
The resource name of the server to be converted to a security domain. (String)

-securityDomain
The name of the security domain to be created and that will contain the settings from the server level security configuration. (String)

Optional parameters

-securityDomainDescription
Specifies a description for the new security domain. (String)

-deleteServer
Specify `true` to remove the server level security configuration and `false` to leave the server level security configuration. (String)

Batch mode example usage

- Using Jython:

```
wsadmin> AdminTask.convertServerSecurityToSecurityDomain ('[serverResource Cell=:Node=myNode:Server=server1
                                        -securityDomain secDomain1 -securityDomainDescription "Migrated from
                                         server security configuration"
                                        -deleteServer true ]'))
```

- Using Jacl:

```
wsadmin> $AdminTask convertServerSecurityToSecurityDomain {-serverResource Cell=:Node=myNode:Server=server1
                                         -securityDomain secDomain1 -securityDomainDescription "Migrated from
                                          server security configuration"
                                         -deleteServer true  }
```

copySecurityDomain

The copySecurityDomain command creates a new security domain by copying an existing security domain. If the security configuration defines an active user registry, provide a realm name for the newly create security domain. If you do not specify a realm name, the system creates a realm name.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the new security domain that the system creates by copying another security domain. (String)

-copyFromSecurityDomainName
Specifies the name of the existing security domain that the system uses to create the new security domain. (String)

Optional parameters

-securityDomainDescription
Specifies a description for the new security domain. (String)

-realmName
Specifies the name of the realm in the new security domain. The system creates a name for the realm if you do not specify a value for this parameter. (String)

Return value

The command returns the configuration ID of the new security domain.

Batch mode example usage

- Using Jython string:

```
AdminTask.copySecurityDomain('-securityDomainName copyOfDomain2 -copyFromSecurityDomainName Domain2')
```

- Using Jython list:

```
AdminTask.copySecurityDomain('-securityDomainName', 'copyOfDomain2', '-copyFromSecurityDomainName', 'Domain2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.copySecurityDomain('-interactive')
```

copySecurityDomainFromGlobalSecurity

The copySecurityDomainFromGlobalSecurity command creates a security domain by copying the global security configuration. If an active user registry exists for the global security configuration, provide a realm name for the newly created security domain. If you do not specify a realm name, then the system creates a realm name.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the new security domain that the system copies from the global security configuration. (String)

Optional parameters

-securityDomainDescription
Specifies a description for the new security domain. (String)

-realmName
Specifies the name of the realm in the new security configuration. The system creates a name for the realm if you do not specify a value for the -realmName parameter. (String)

Return value

The command returns the configuration ID of the new security domain.

Batch mode example usage

- Using Jython string:

```
AdminTask.copySecurityDomainFromGlobalSecurity('-securityDomainName GSCopy -securityDomainDescription
"copy of global security" -realmName myRealm')
```

- Using Jython list:

```
AdminTask.copySecurityDomainFromGlobalSecurity('-securityDomainName', 'GSCopy', '-securityDomainDescription',
'"copy of global security"', '-realmName myRealm')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.copySecurityDomainFromGlobalSecurity('-interactive')
```

createSecurityDomain

The createSecurityDomain command creates the security domain-security.xml and domain-security-map.xml files under the profile_root/config/cells/*cellName*/securityDomain/*configurationName* directory. The system creates an empty domain-security.xml file.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the new security domain to create. (String)

Optional parameters

-securityDomainDescription
Specifies a description of the new security domain. (String)

Return value

The command returns the configuration ID of the new security domain.

Batch mode example usage

- Using Jython string:

```
AdminTask.createSecurityDomain('-securityDomainName newDomain -securityDomainDescription "new security domain"')
```

- Using Jython list:

```
AdminTask.createSecurityDomain('-securityDomainName', 'newDomain', '-securityDomainDescription',
'"new security domain"')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createSecurityDomain('-interactive')
```

deleteSecurityDomain

The deleteSecurityDomain command removes the domain-security.xml and domain-security-map.xml files from the security domain directory. The command returns an error if resources are mapped to the security domain of interest. To delete the security domain when resources are mapped to the security domain of interest, specify the value for the -force parameter as `true`.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain to delete. (String)

Optional parameters

-force
Specifies that the system deletes the security domain without checking for resources that are associated with the domain. Use this option when the resources in the security domains are not valid resources. The default value for the -force parameter is `false`. (Boolean)

Return value

The command does not return output if the system successfully removes the security domain configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteSecurityDomain('-securityDomainName mySecurityDomain -force true')
```

- Using Jython list:

```
AdminTask.deleteSecurityDomain('-securityDomainName', 'mySecurityDomain', '-force', 'true')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSecurityDomain('-interactive')
```

getSecurityDomainForResource

The getSecurityDomainForResource command displays the security domain for a specific resource. If the resource is not mapped to a domain, the command does not return output.

Target object

None.

Required parameters

-resourceName
Specifies the name of the resource of interest. Specify the value in the following format: `Cell=:Node=myNode:Server=myServer` (String)

Optional parameters

-getEffectiveDomain
Specifies whether the command returns the effective domain of the resource if the resource is not

directly mapped to a domain. The default value is `true`. Specify `false` if you do not want to display the effective domain if the resource is not directly mapped to a domain. (Boolean)

Return value

The command returns the security domain name as a string.

Batch mode example usage

- Using Jython string:

```
AdminTask.getSecurityDomainForResource('-resourceName Cell=:Node=myNode:Server=myServer')
```

- Using Jython list:

```
AdminTask.getSecurityDomainForResource('-resourceName', 'Cell=:Node=myNode:Server=myServer')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getSecurityDomainForResource('-interactive')
```

listResourcesInSecurityDomain

The listResourcesInSecurityDomain command displays the servers or clusters that are associated with a specific security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain of interest. (String)

-expandCell
Specifies whether to display the servers in the cell. Specify `true` to display the specific servers, or specify `false` to list the cell information only. (Boolean)

Return value

The command returns an array that contains the names of the resources that are mapped to the security domain of interest in the format: `Cell=cell name:Node=node name:Server=server name`.

Batch mode example usage

- Using Jython string:

```
AdminTask.listResourcesInSecurityDomain('-securityDomainName myDomain')
```

- Using Jython list:

```
AdminTask.listResourcesInSecurityDomain('-securityDomainName', 'myDomain')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listResourcesInSecurityDomain('-interactive')
```

listSecurityDomains

The listSecurityDomains command lists each security domain configured for the server.

Target object

None.

Optional parameters

-listDescription
Specifies whether to display the description of the security domains. Specify `true` to display the descriptions of the security domains. (Boolean)

-doNotDisplaySpecialDomains
Specifies whether to exclude special domains. Specify `true` to exclude the special domains in the command output, or `false` to display the special domains. (Boolean)

Return value

The command returns an array that contains the names of security domains that are configured for the server. The command returns an array of attribute lists that contain the name and description for each security domain if the -listDescription parameter is specified.

Batch mode example usage

- Using Jython string:

```
AdminTask.listSecurityDomains('-listDescription true')
```

- Using Jython list:

```
AdminTask.listSecurityDomains('-listDescription', 'true')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSecurityDomains('-interactive')
```

listSecurityDomainsForResources

The listSecurityDomainsForResources command lists the security domains that are associated with the resources of interest.

Target object

None.

Required parameters

-resourceNames
Specifies one or more resources for which the command returns the associated security domains. Specify each resource separated by the plus sign character (+). (String)

Return value

The command returns the list of resources specified by the -resourceNames parameter and the security domains to which each resource is mapped.

Batch mode example usage

- Using Jython string:

```
AdminTask.listSecurityDomainsForResources('-resourceNames resource1+resource2+resource3')
```

- Using Jython list:

```
AdminTask.listSecurityDomainsForResources('-resourceNames', 'resource1+resource2+resource3')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSecurityDomainsForResources('-interactive')
```

mapResourceToSecurityDomain

The mapResourceToSecurityDomain command maps a resource to a security domain. The system adds an entry for each resource to the domain-security-map.xml file.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain of interest. (String)

-resourceName
Specifies the name of the resource to which the system maps the security domain of interest. Specify the value in the following format: `Cell=:Node=`*`myNode`*`:Server=`*`myServer`* (String)

Return value

The command does not return output if the system successfully assigns the resource to the security domain of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.mapResourceToSecurityDomain('-securityDomainName mySecurityDomain -resourceName
-resourceName Cell=:Node=myNode:Server=myServer')
```

- Using Jython list:

```
AdminTask.mapResourceToSecurityDomain('-securityDomainName', 'mySecurityDomain', '-resourceName',
'-resourceName Cell=:Node=myNode:Server=myServer')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.mapResourceToSecurityDomain('-interactive')
```

modifySecurityDomain

The modifySecurityDomain command changes the description of a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain to edit. (String)

Optional parameters

-securityDomainDescription
Specifies the new description for the security domain of interest. (String)

Return value

The command does not return output if the system successfully modifies the security domain.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifySecurityDomain('-securityDomainName myDomain -securityDomainDescription
"my new description"')
```

- Using Jython list:

```
AdminTask.modifySecurityDomain('-securityDomainName', 'myDomain', '-securityDomainDescription',=
'"my new description"')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifySecurityDomain('-interactive')
```

removeResourceFromSecurityDomain

The removeResourceFromSecurityDomain command removes a resource from a security domain mapping. The command removes the resource entry from the domain-security-map.xml file.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain from which to remove the resource. (String)

-resourceName
Specifies the name of the resource to remove. Specify the value in the following format: `Cell=:Node=myNode:Server=myServer` (String)

Return value

The command does not return output if the system successfully removes the resource from the security domain.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeResourceFromSecurityDomain('-securityDomainName myDomain -resourceName
Cell=:Node=myNode:Server=myServer')
```

- Using Jython list:

```
AdminTask.removeResourceFromSecurityDomain('-securityDomainName', 'myDomain', '-resourceName',
'Cell=:Node=myNode:Server=myServer')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeResourceFromSecurityDomain('-interactive')
```

SecurityConfigurationCommands command group for the AdminTask object

You can use the Jython scripting language to configure security with the wsadmin tool. Use the commands and parameters in the SecurityConfigurationCommands group to configure and manage user registries, single sign-on, data entries, trust association, login modules, and interceptors.

Use the following command to administer user registry configurations:

- configureAdminCustomUserRegistry
- configureAdminLDAPUserRegistry
- configureAdminLocalOSUserRegistry

- configureAdminWIMUserRegistry
- configureAppCustomUserRegistry
- configureAppLDAPUserRegistry
- configureAppLocalOSUserRegistry
- configureAppWIMUserRegistry
- getLTPATimeout
- setLTPATimeout
- getUserRegistryInfo
- unconfigureUserRegistry

Use the following commands to administer Java Authentication and Authorization Service (JAAS) login configurations:
- configureLoginEntry
- configureLoginModule
- getJAASLoginEntryInfo
- listJAASLoginEntries
- listLoginModules
- unconfigureJAASLoginEntry
- unconfigureLoginModule

Use the following commands to administer data entry configurations:
- createAuthDataEntry
- deleteAuthDataEntry
- getAuthDataEntry
- listAuthDataEntries
- modifyAuthDataEntry

Use the following commands to administer Common Secure Interoperability Version 2 (CSIv2) configurations:
- configureCSIInbound
- configureCSIOutbound
- getCSIInboundInfo
- getCSIOutboundInfo
- unconfigureCSIInbound
- unconfigureCSIOutbound

Use the following commands to administer trust association configurations:
- configureInterceptor
- configureTrustAssociation
- getTrustAssociationInfo
- listInterceptors
- unconfigureInterceptor
- unconfigureTrustAssociation

Use the following commands to manage your security configuration:
- “applyWizardSettings” on page 150
- configureAuthzConfig

- configureSingleSignon
- getActiveSecuritySettings
- “getAuthzConfigInfo” on page 153
- “getSingleSignon” on page 154
- setAdminActiveSecuritySettings
- setAppActiveSecuritySettings
- unconfigureAuthzConfig
- unsetAppActiveSecuritySettings

configureAdminCustomUserRegistry

The configureAdminCustomUserRegistry command configures a custom user registry in the global security configuration.

Note: This command is not supported in a local mode.

Target object

None.

Optional parameters

-autoGenerateServerId
Specifies whether the command automatically generates the server identity that the system uses for internal process communication. Specify `true` to automatically generate the server identity. (Boolean)

-serverId
Specifies the server identity in the repository that the system uses for internal process communication. (String)

-serverIdPassword
Specifies the password that corresponds to the server identity. (String)

-primaryAdminId
Specifies the name of the user with administrative privileges that is defined in the registry. This parameter does not apply to security configurations. (String)

-customRegClass
Specifies the class name that implements the UserRegistry interface in com.ibm.websphere.security property. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAdminCustomUserRegistry('-autoGenerateServerId true -serverIdPassword password4server
-primaryAdminId serverAdmin')
```

- Using Jython list:

```
AdminTask.configureAdminCustomUserRegistry(['autoGenerateServerId', 'true', '-serverIdPassword', 'password4server',
'-primaryAdminId', 'serverAdmin'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAdminCustomUserRegistry('-interactive')
```

configureAdminLDAPUserRegistry

The configureAdminLDAPUserRegistry command configures a Lightweight Directory Access Protocol (LDAP) user registry in the global security configuration.

Note: This command is not supported in a local mode.

Target object

None.

Optional parameters

-autoGenerateServerId
Specifies whether the command automatically generates the server identity used for internal process communication. Specify `true` to automatically generate the server identity. (Boolean)

-serverId
Specifies the server identity in the repository that the system uses for internal process communication. (String)

-serverIdPassword
Specifies the password that corresponds to the server identity. (String)

-primaryAdminId
Specifies the name of the user with administrative privileges that is defined in the registry. This parameter does not apply to security configurations. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-ldapServerType
Specifies the type of LDAP server. The default type is `IBM_DIRECTORY_SERVER`. (String)

Specify one of the following valid values:

- `IBM_DIRECTORY_SERVER`
- `IPLANET`
- `NETSCAPE`
- `NDS`
- `DOMINO502`
- `SECUREWAY`
- `ACTIVE_DIRECTORY`
- `CUSTOM`

-ldapHost
Specifies the host name of the LDAP server. (String)

-ldapPort
Specifies the port that the system uses to access the LDAP server. The default value is `389`. (String)

-baseDN
Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed. (String)

-bindDN
Specifies the distinguished name for the application server, which is used to bind to the directory service. (String)

-bindPassword
Specifies the binding DN password for the LDAP server. (String)

-searchTimeout
Specifies the timeout value in seconds for an LDAP server to respond before stopping a request. The default value is `120` seconds. (Long)

-reuseConnection
Specifies whether the server reuses the LDAP connection. By default, this option is enabled. Specify `false` for this parameter only in rare situations where a router is used to distribute requests to multiple LDAP servers and when the router does not support affinity. (Boolean)

Note: When you disable the reuse of the LDAP connection, the application server creates a new LDAP connection for every LDAP search request. This situation impacts system performance if your environment requires extensive LDAP calls. This option is provided because the router is not sending the request to the same LDAP server. The option is also used when the idle connection timeout value or firewall timeout value between the application server and LDAP is too small.

-userFilter
Specifies the LDAP filter clause that the system uses to search the user registry for users. The default value is the default user filter for the LDAP server type. (String)

-groupFilter
Specifies the LDAP filter clause that the system uses to search the user registry for groups. The default value is the default group filter for the LDAP server type. (String)

-userIdMap
Specifies the LDAP filter that maps the short name of a user to an LDAP entry. The default value is the default user filter for the LDAP server type. (String)

-groupIdMap
Specifies the LDAP filter that maps the short name of a group to an LDAP entry. The default value is the default group filter for the LDAP server type. (String)

-groupMemberIdMap
Specifies the LDAP filter that identifies users to group memberships. (String)

-certificateMapMode
Specifies whether to map X.509 certificates into an LDAP directory by `EXACT_DN` or `CERTIFICATE_FILTER`. Specify `CERTIFICATE_FILTER` to use the specified certificate filter for the mapping. (String)

-certificateFilter
Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP registry. (String)

The syntax or structure of this filter is: `(&(uid=${SubjectCN})(objectclass=inetOrgPerson))`. The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- ${UniqueKey}
- ${PublicKey}
- ${Issuer}
- ${NotAfter}
- ${NotBefore}
- ${SerialNumber}
- ${SigAlgName}
- ${SigAlgOID}
- ${SigAlgParams}
- ${SubjectCN}
- ${Version}

-krbUserFilter
Specifies that the default value is the default user filter for the LDAP server type. (String)

-nestedGroupSearch
Specifies whether to perform a recursive nested group search. Specify `true` to perform a recursive nested group search, or specify `false` to disable recursive nested group searching. (Boolean)

-sslEnabled
Specifies whether to enable Secure Sockets Layer (SSL). Specify `true` to enable an SSL connection to the LDAP server. (Boolean)

-sslConfig
Specifies the SSL configuration alias to use for the secure LDAP connection. (String)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

-resetDefaultFilters
Specify `true`, to reset all of the filter values to the default value of the LDAP server type. The default value for this parameter is `false`. The LDAP filter attributes reset are: userFilter, groupFilter, userIdMap, groupIdMap, groupMemberIdMap and krbUserFilter. If any of the other filter flags are used to specify a filter value on the command line at the same time resetDefaultFilter is set to `true`, the filter value specified is used. Any filter not specified on the command line at the time is reset to the default value of the LDAP server type.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAdminCustomUserRegistry('-autoGenerateServerId true -serverIdPassword password4server
-primaryAdminId serverAdmin -ldapServerType NETSCAPE -ldapHost 195.168.1.1')
```

- Using Jython list:

```
AdminTask.configureAdminCustomUserRegistry(['-autoGenerateServerId', 'true', '-serverIdPassword', 'password4server',
'-primaryAdminId', 'serverAdmin', '-ldapServerType', 'NETSCAPE', '-ldapHost', '195.168.1.1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAdminLDAPUserRegistry('-interactive')
```

configureAdminLocalOSUserRegistry

The configureAdminLocalOSUserRegistry command configures a local operating system user registry in the global security configuration.

Note: This command is not supported in a local mode.

Target object

None.

Optional parameters

-autoGenerateServerId
Specifies whether the command automatically generates the server identity used for internal process communication. Specify `true` to automatically generate the server identity. (Boolean)

-serverId
Specifies the server identity in the repository that the system uses for internal process communication. (String)

-serverIdPassword
Specifies the password that corresponds to the server identity. (String)

-primaryAdminId
Specifies the name of the user with administrative privileges that is defined in the registry. This parameter does not apply to security configurations. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAdminLocalOSUserRegistry('-autoGenerateServerId true -serverIdPassword password4server
-primaryAdminId serverAdmin')
```

- Using Jython list:

```
AdminTask.configureAdminLocalOSUserRegistry(['autoGenerateServerId', 'true', '-serverIdPassword', 'password4server',
'-primaryAdminId', 'serverAdmin'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAdminLocalOSUserRegistry('-interactive')
```

configureAdminWIMUserRegistry

The configureAdminWIMUserRegistry command configures a federated repository user registry in the administrative security configuration.

Note: This command is not supported in a local mode.

Target object

None.

Optional parameters

-autoGenerateServerId
Specifies whether the command automatically generates the server identity used for internal process communication. Specify `true` to automatically generate the server identity. (Boolean)

-serverId
Specifies the server identity in the repository that the system uses for internal process communication. (String)

-serverIdPassword
Specifies the password that corresponds to the server identity. (String)

-primaryAdminId
Specifies the name of the user with administrative privileges that is defined in the registry. This parameter does not apply to security configurations. (String)

-realmName
Specifies the realm of the user registry. The system automatically generates a realm name if you do not specify a value for the -realmName parameter. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAdminWIMUserRegistry('-autoGenerateServerId true -serverIdPassword password4server
-primaryAdminId serverAdmin')
```

- Using Jython list:

```
AdminTask.configureAdminWIMUserRegistry(['autoGenerateServerId', 'true', '-serverIdPassword', 'password4server',
'-primaryAdminId', 'serverAdmin'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAdminWIMUserRegistry('-interactive')
```

configureAppCustomUserRegistry

The configureAppCustomUserRegistry command configures a custom user registry in an application security domain.

Note: This command is not supported in a local mode.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Optional parameters

-realmName
Specifies the realm of the user registry. The system automatically generates a realm name if you do not specify a value for the -realmName parameter. (String)

-customRegClass
Specifies the class name that implements the UserRegistry interface in com.ibm.websphere.security property. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAppCustomUserRegistry('-securityDomainName testDomain -realmName server_name.domain:port_number')
```

- Using Jython list:

```
AdminTask.configureAppCustomUserRegistry(['-securityDomainName', 'testDomain', '-realmName',
'server_name.domain:port_number'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAppCustomUserRegistry('-interactive')
```

configureAppLDAPUserRegistry

The configureAppLDAPUserRegistry command configures LDAP user registries in a security configuration or a global security configuration.

Note: This command is not supported in a local mode.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Optional parameters

-realmName
Specifies the realm of the user registry. The system automatically generates a realm name if you do not specify a value for the -realmName parameter. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-ldapServerType
Specifies the type of LDAP server. The default type is `IBM_DIRECTORY_SERVER`. (String)

Specify one of the following valid values:

- `IBM_DIRECTORY_SERVER`
- `IPLANET`
- `NETSCAPE`
- `NDS`
- `DOMINO502`
- `SECUREWAY`
- `ACTIVE_DIRECTORY`
- `CUSTOM`

-ldapHost
Specifies the host name of the LDAP server. (String)

-ldapPort
Specifies the port that the system uses to access the LDAP server. The default value is `389`. (String)

-baseDN
Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed. (String)

-bindDN
Specifies the distinguished name for the application server, which is used to bind to the directory service. (String)

-bindPassword
Specifies the binding DN password for the LDAP server. (String)

-searchTimeout
Specifies the timeout value in seconds for an LDAP server to respond before stopping a request. The default value is `120` seconds. (Long Integer)

-reuseConnection
Specifies whether the server reuses the LDAP connection. By default, this option is enabled. Specify `false` for this parameter only in rare situations where a router is used to distribute requests to multiple LDAP servers and when the router does not support affinity. (Boolean)

Note: When you disable the reuse of the LDAP connection, the application server creates a new LDAP connection for every LDAP search request. This situation impacts system performance if your environment requires extensive LDAP calls. This option is provided because the router is not sending the request to the same LDAP server. The option is also used when the idle connection timeout value or firewall timeout value between the application server and LDAP is too small.

-userFilter
Specifies the LDAP filter clause that the system uses to search the user registry for users. The default value is the default user filter for the LDAP server type. (String)

-groupFilter
Specifies the LDAP filter clause that the system uses to search the user registry for groups. The default value is the default group filter for the LDAP server type. (String)

-userIdMap
Specifies the LDAP filter that maps the short name of a user to an LDAP entry. The default value is the default user filter for the LDAP server type. (String)

-groupIdMap
Specifies the LDAP filter that maps the short name of a group to an LDAP entry. The default value is the default group filter for the LDAP server type. (String)

-groupMemberIdMap
Specifies the LDAP filter that identifies users to group memberships. (String)

-certificateMapMode
Specifies whether to map X.509 certificates into an LDAP directory by `EXACT_DN` or `CERTIFICATE_FILTER`. Specify `CERTIFICATE_FILTER` to use the specified certificate filter for the mapping. (String)

-certificateFilter
Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP registry. (String)

The syntax or structure of this filter is: `(&(uid=${SubjectCN})(objectclass=inetOrgPerson))`. The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- ${UniqueKey}
- ${PublicKey}
- ${Issuer}
- ${NotAfter}
- ${NotBefore}
- ${SerialNumber}
- ${SigAlgName}
- ${SigAlgOID}
- ${SigAlgParams}
- ${SubjectCN}
- ${Version}

-krbUserFilter
Specifies the default value is the default user filter for the LDAP server type. (String)

-nestedGroupSearch
Specifies whether to perform a recursive nested group search. Specify `true` to perform a recursive nested group search, or specify `false` to disable recursive nested group searching. (Boolean)

-sslEnabled
Specifies whether to enable Secure Sockets Layer (SSL). Specify `true` to enable an SSL connection to the LDAP server. (Boolean)

-sslConfig
Specifies the SSL configuration alias to use for the secure LDAP connection. (String)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

-resetDefaultFilters
Specify `true`, to reset all of the filter values to the default value of the LDAP server type. The default value for this parameter is `false`. The LDAP filter attributes reset are: userFilter, groupFilter, userIdMap, groupIdMap, groupMemberIdMap and krbUserFilter. If any of the other filter flags are used to specify a filter value on the command line at the same time resetDefaultFilter is set to `true`, the filter value specified is used. Any filter not specified on the command line at the time is reset to the default value of the LDAP server type.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAppLDAPUserRegistry('-securityDomainName testDomain -ldapServerType NETSCAPE -ldapHost
195.168.1.1 -searchTimeout 300')
```

- Using Jython list:

```
AdminTask.configureAppLDAPUserRegistry(['-securityDomainName', 'testDomain', '-ldapServerType',
'NETSCAPE', '-ldapHost', '195.168.1.1', '-searchTimeout', '300'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAppLDAPUserRegistry('-interactive')
```

configureAppLocalOSUserRegistry

The configureAppLocalOSUserRegistry command configures a local operating system user registry in a security domain.

Note: This command is not supported in a local mode.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Optional parameters

-realmName
Specifies the realm of the user registry. The system automatically generates a realm name if you do not specify a value for the -realmName parameter. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAppLocalOSUserRegistry('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.configureAppLocalOSUserRegistry(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAppLocalOSUserRegistry('-interactive')
```

configureAppWIMUserRegistry

The configureAppWIMUserRegistry command configures federated repository user registries in a security domain.

Note: This command is not supported in a local mode.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Optional parameters

-realmName
Specifies the realm of the user registry. The system automatically generates a realm name if you do not specify a value for the -realmName parameter. (String)

-verifyRegistry
Specifies whether to verify that the user registry configuration is correct. If you set this parameter to `true`, then the system verifies the registry by making a call to the user registry to verify the admin ID. If you specify a server ID and password, then the system verifies the user and password with the user

registry. Set the parameter to `false` to store the attributes in the configuration without validation. The command verifies the registry configuration by default. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

-useGlobalFederatedRepository
Specifies whether to use the same instance of federated repository for the domain as is defined in the global domain. Specify `true` to use the same instance as defined in the global domain. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAppWIMUserRegistry('-securityDomainName testDomain -realmName testRealm')
```

- Using Jython list:

```
AdminTask.configureAppWIMUserRegistry(['securityDomainName', 'testDomain', '-realmName', 'testRealm'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAppWIMUserRegistry('-interactive')
```

getLTPATimeout

The getLTPATimeout command displays the number of seconds that the system waits before the LTPA request reaches timeout.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns the number of seconds that the server waits before the LTPA request is cancelled.

Batch mode example usage

- Using Jython string:

```
AdminTask.getLTPATimeout('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getLTPATimeout(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getLTPATimeout('-interactive')
```

setLTPATimeout

The setLTPATimeout command sets the amount of time that the system waits before the LTPA request becomes invalid.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-timeout
Specifies the amount of time, in seconds, before the request times out. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.setLTPATimeout('-timeout 120')
```

- Using Jython list:

```
AdminTask.setLTPATimeout(['timeout', '120'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setLTPATimeout('-interactive')
```

getUserRegistryInfo

The getUserRegistryInfo command displays information about a user registry in a security domain or in the global security configuration. If you do not specify a value for the -userRegistryType parameter, the command returns the active user registry information.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-userRegistryType
Specifies the type of user registry. Specify `LDAPUserRegistry` for LDAP user registries. Specify `WIMUserRegistry` for federated repository user registries. Specify `CustomUserRegistry` for custom user registries. Specify `LocalOSUserRegisty` for local operating system user registries. (String)

Return value

The command returns configuration information in the form of attribute and value pairs for the user registry object of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.getUserRegistryInfo('-securityDomainName testDomain -userRegistryType LDAPUserRegistry')
```

- Using Jython list:

```
AdminTask.getUserRegistryInfo(['securityDomainName', 'testDomain', '-userRegistryType', 'LDAPUserRegistry'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getUserRegistryInfo('-interactive')
```

unconfigureUserRegistry

The unconfigureUserRegistry command modifies the user registry. For a global security configuration, the command reduces the user registry to the minimum registry values. For application-level security, the command removes the user registry from the security domain of interest.

Target object

None.

Required parameters

-userRegistryType
Specifies the type of user registry. Specify `LDAPUserRegistry` for LDAP user registries. Specify `WIMUserRegistry` for federated repository user registries. Specify `CustomUserRegistry` for custom user registries. Specify `LocalOSUserRegisty` for local operating system user registries. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureUserRegistry('-userRegistryType WIMUserRegistry -securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureUserRegistry(['userRegistryType', 'WIMUserRegistry', '-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureUserRegistry('-interactive')
```

configureJAASLoginEntry

The configureJAASLoginEntry command configures a Java Authentication and Authorization Service (JAAS) login entry in a security domain or in the global security configuration. You can use this command to modify existing JAAS login entries or to create new login entries.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. If you do not specify a security domain name, the system updates the global security configuration. (String)

-loginModules
Specifies a comma (**,**) separated list of login module class names. Specify the list in the order that the system calls them. (String)

-authStrategies
Specifies a comma-separated list of authentication strategies that sets the authentication behavior as authentication proceeds down the list of login modules. You must specify one authentication strategy for each login module. (String)

Specify one or many of the following values in a comma (**,**) separated list:

- `REQUIRED`

  Specifies that the LoginModule module is required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list for each realm.

- `REQUISITE`

  Specifies that the LoginModule module is required to succeed. If authentication is successful, the process continues down the LoginModule list in the realm entry. If authentication fails, control immediately returns to the application. Authentication does not proceed down the LoginModule list.

- `SUFFICIENT`

  Specifies that the LoginModule module is not required to succeed. If authentication succeeds, control immediately returns to the application. Authentication does not proceed down the LoginModule list. If authentication fails, the process continues down the list.

- `OPTIONAL`

  Specifies that the LoginModule module is not required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureJAASLoginEntry('[-loginType application -loginEntryAlias JAASLoginEntry1 -authStrategies "
REQUIRED,REQUISITE"]')
```

- Using Jython list:

```
AdminTask.configureJAASLoginEntry(['loginType', 'application', '-loginEntryAlias', 'JAASLoginEntry1', '-authStrategies',
'REQUIRED,REQUISITE'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureJAASLoginEntry('-interactive')
```

configureLoginModule

The configureLoginModule command modifies an existing login module or creates a new login module on an existing JAAS login entry in the global security configuration or in a security domain.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

-loginModule
Specifies the name of the login module. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. (String)

-useLoginModuleProxy
Specifies that the JAAS loads the login module proxy class. JAAS then delegates calls to the login module classes that are defined in the Module class name field. Specify `true` to use the login module proxy. (Boolean)

-authStrategy
Specifies the authentication behavior as authentication proceeds down the list of login modules. (String)

Specify one of the following values:

- `REQUIRED`

  Specifies that the LoginModule module is required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list for each realm.

- `REQUISITE`

  Specifies that the LoginModule module is required to succeed. If authentication is successful, the process continues down the LoginModule list in the realm entry. If authentication fails, control immediately returns to the application. Authentication does not proceed down the LoginModule list.

- `SUFFICIENT`

  Specifies that the LoginModule module is not required to succeed. If authentication succeeds, control immediately returns to the application. Authentication does not proceed down the LoginModule list. If authentication fails, the process continues down the list.

- `OPTIONAL`

  Specifies that the LoginModule module is not required to succeed. Whether authentication succeeds or fails, the process still continues down the LoginModule list.

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `["attr1=value1","attr2=value2"]` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureLoginModule('-loginType application -loginEntryAlias JAASLoginEntry1 -loginModule class1')
```

- Using Jython list:

```
AdminTask.configureLoginModule(['loginType', 'application', '-loginEntryAlias', 'JAASLoginEntry1', '-loginModule',
'class1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureLoginModule('-interactive')
```

getJAASLoginEntryInfo

The getJAASLoginEntryInfo command displays configuration for a specific JAAS login entry.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an attribute list that contains configuration information for the JAAS login entry of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.getJAASLoginEntryInfo('-loginType application -loginEntryAlias JAASLoginEntry -securityDomainName
testDomain')
```

- Using Jython list:

```
AdminTask.getJAASLoginEntryInfo(['loginType', 'application', '-loginEntryAlias', 'JAASLoginEntry', '-securityDomainName',
'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getJAASLoginEntryInfo('-interactive')
```

listJAASLoginEntries

The listJAASLoginEntries command displays each defined JAAS login modules for given type in a security domain or the global security configuration.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an array of attribute lists that contain the login entries for the login type of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.listJAASLoginEntries('-loginType application -securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.listJAASLoginEntries(['loginType', 'application','-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listJAASLoginEntries('-interactive')
```

listLoginModules

The listLoginModules command displays the class names and associated options for a specific JAAS login module in a security domain or in the global security configuration.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an array that contains the login modules in a specific login entry.

Batch mode example usage

- Using Jython string:

```
AdminTask.listLoginModules('-loginType system -loginEntryAlias JAASLoginEntry')
```

- Using Jython list:

```
AdminTask.listLoginModules(['loginType', 'system', '-loginEntryAlias', 'JAASLoginEntry'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listLoginModules('-interactive')
```

unconfigureJAASLoginEntry

The unconfigureJAASLoginEntry command removes a JAAS login entry from the global security configuration or a security domain. You cannot remove all login entries. The command returns an error if it cannot remove the login entry of interest.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureJAASLoginEntry('-loginType application -loginEntryAlias myLoginEntry')
```

- Using Jython list:

```
AdminTask.unconfigureJAASLoginEntry(['loginType', 'application', '-loginEntryAlias', 'myLoginEntry'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureJAASLoginEntry('-interactive')
```

unconfigureLoginModule

The unconfigureLoginModule command removes a login module class from a login module entry.

Target object

None.

Required parameters

-loginType
Specifies the type of JAAS login entry of interest. Specify `system` for the system login type or `application` for the application login type. (String)

-loginEntryAlias
Specifies an alias that identifies the JAAS login entry in the configuration. (String)

-loginModule
Specifies the name of the login module class to remove from the configuration. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureLoginModule('-loginType system -loginEntryAlias systemLoginEntry -loginModule moduleClass')
```

- Using Jython list:

```
AdminTask.unconfigureLoginModule(['loginType', 'system', '-loginEntryAlias', 'systemLoginEntry', '-loginModule',
'moduleClass'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureLoginModule('-interactive')
```

createAuthDataEntry

The createAuthDataEntry command creates an authentication data entry for a J2EE Connector architecture (J2C) connector in the global security or security domain configuration.

Target object

None.

Required parameters

-alias
Specifies the name that uniquely identifies the authentication data entry. (String)

-user
Specifies the J2C authentication data user ID. (String)

-password
Specifies the password to use for the target enterprise information system (EIS). (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain configuration. The application server uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-description
Specifies a description of the authentication data entry. (String)

Return value

The command returns the object name of the new authentication data entry object.

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuthDataEntry('-alias dataEntry1 -user userID -password userIDpw')
```

- Using Jython list:

```
AdminTask.createAuthDataEntry(['alias', 'dataEntry1', '-user', 'userID', '-password', 'userIDpw'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuthDataEntry('-interactive')
```

deleteAuthDataEntry

The deleteAuthDataEntry command removes an authentication data entry for a J2C connector in a global security or security domain configuration.

Target object

None.

Required parameters

-alias
Specifies the name that uniquely identifies the authentication data entry. (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain configuration. The application server uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuthDataEntry('-alias dataEntry1')
```

- Using Jython list:

```
AdminTask.deleteAuthDataEntry(['alias', 'dataEntry1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuthDataEntry('-interactive')
```

getAuthDataEntry

The getAuthDataEntry command displays information about an authentication data entry for the J2C connector in the global security configuration or for a specific security domain.

Target object

None.

Required parameters

-alias
Specifies the name that uniquely identifies the authentication data entry. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an attribute list that contains the authentication data entry attributes and values.

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuthDataEntry('-alias authDataEntry1 -securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getAuthDataEntry(['alias', 'authDataEntry1', '-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuthDataEntry('-interactive')
```

listAuthDataEntries

The listAuthDataEntries command displays each authentication data entry in the global security configuration or in a security domain.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an array of attribute lists for each authentication data entry.

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuthDataEntries('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.listAuthDataEntries(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuthDataEntries('-interactive')
```

modifyAuthDataEntry

The modifyAuthDataEntry command modifies an authentication data entry for a J2C connector in the global security or security domain configuration.

Target object

None.

Required parameters

-alias
Specifies the name that uniquely identifies the authentication data entry. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-user
Specifies the J2C authentication data user ID. (String)

-password
Specifies the password to use for the target enterprise information system (EIS). (String)

-description
Specifies a description for the authentication data entry. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuthDataEntry('-alias dataEntry1 -user userID1 -password newPassword')
```

- Using Jython list:

```
AdminTask.modifyAuthDataEntry(['alias', 'dataEntry1', '-user', 'userID1', '-password', 'newPassword'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuthDataEntry('-interactive')
```

configureCSIInbound

The configureCSIInbound command configures CSIv2 inbound authentication on a security domain or on the global security configuration. When configuring CSI inbound authentication in a security domain for the first time, the CSI objects are copied from global security so that any changes to that configuration are applied.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. If one is not provided the task will work on the global security user registry configuration. (String)

-messageLevelAuth
Specifies whether clients connecting to this server must specify a user ID and password. Specify `Never` to disable the user ID and password requirement. Specify `Supported` to accept a user ID and password. Specify `Required` to require a user ID and password. (String)

-supportedAuthMechList
Specifies the authentication mechanism to use. Specify `KRB5` for Kerberos authentication, `LTPA` for Lightweight Third-Party Authentication, `BasicAuth` for BasicAuth authentication, and `custom` to use your own authentication token implementation. You can specify more then one in a space-separated list. (String)

-clientCertAuth
Specifies whether a client that connects to the server must connect using an SSL certificate. Specify `Never` to allow clients to connect without SSL certificates. Specify `Supported` to accept clients connecting with and without SSL certificates. Specify `Required` to require clients to use SSL certificate. (String)

-transportLayer
Specifies the transport layer support level. Specify `Never` to disable transport layer support. Specify `Supported` to enable transport layer support. Specify `Required` to require transport layer support. (String)

-sslConfiguration
Specifies the SSL configuration alias to use for inbound transport. (String)

-enableIdentityAssertion
Specifies whether to enable identity assertion. When using the identity assertion authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element. Specify `true` for the -enableIdentityAssertion parameter to enable identity assertion. (Boolean)

-trustedIdentities
Specifies a list of trusted server identities, separated by the pipe character (|). To specify a null value, set the value of the -trustedIdentities parameter as an empty string (""). (String)

-statefulSession
Specifies whether to enable a stateful session. Specify `true` to enable a stateful session. (Boolean)

-enableAttributePropagation
Specifies whether to enable security attribute propagation. Security attribute propagation allows the application server to transport authenticated Subject contents and security context information from one server to another in your configuration. Specify `true` to enable security attribute propagation. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureCSIInbound('[-securityDomainName testDomain -messageLevelAuth Required
-supportedAuthMechList "KRB5 LTPA"]')
```

- Using Jython list:

```
AdminTask.configureCSIInbound(['-securityDomainName', 'testDomain', '-messageLevelAuth', 'Required',
'-supportedAuthMechList', 'KRB5 LTPA'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureCSIInbound('-interactive')
```

configureCSIOutbound

The configureCSIOutbound command configures the CSIv2 outbound authentication in a security domain or in the global security configuration. When configuring CSI Outbound in a security domain for the first time, the application server copies the CSI objects from global security. Then, the application server applies the changes to that configuration from the command.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. (String)

-enableAttributePropagation
Specifies whether to enable security attribute propagation. Security attribute propagation allows the application server to transport authenticated Subject contents and security context information from one server to another in your configuration. Specify `true` to enable security attribute propagation. (Boolean)

-enableIdentityAssertion
Specifies whether to enable identity assertion. When using the identity assertion authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element. Specify `true` for the -enableIdentityAssertion parameter to enable identity assertion. (Boolean)

-useServerIdentity
Specifies whether to use the server identity to establish trust with the target server. Specify `true` to use the server identity. (Boolean)

-trustedId
Specifies the trusted identity that the application server uses to establish trust with the target server. (String)

-trustedIdentityPassword
Specifies the password of the trusted server identity. (String)

-messageLevelAuth
Specifies whether clients connecting to this server must specify a user ID and password. Specify `includeNever` to disable the user ID and password requirement. Specify `Supported` to accept a user ID and password. Specify `Required` to require a user ID and password. (String)

-supportedAuthMechList
Specifies the authentication mechanism to use. Specify `KRB5` for Kerberos authentication, `LTPA` for Lightweight Third-Party Authentication, `BasicAuth` for BasicAuth authentication, and `custom` to use your own authentication token implementation. You can specify more then one in a space-separated list. (String)

-clientCertAuth
Specifies whether a client that connects to the server must connect using an SSL certificate. Specify `Never` to allow clients to connect without SSL certificates. Specify `Supported` to accept clients connecting with and without SSL certificates. Specify `Required` to require clients to use SSL certificate. (String)

-transportLayer
Specifies the transport layer support level. Specify `Never` to disable transport layer support. Specify `Supported` to enable transport layer support. Specify `Required` to require transport layer support. (String)

-sslConfiguration
Specifies the SSL configuration alias to use for inbound transport. (String)

-statefulSession
Specifies whether to enable a stateful session. Specify `true` to enable a stateful session. (Boolean)

-enableOutboundMapping
Specifies whether to enable custom outbound identity mapping. Specify `true` to enable custom outbound identity mapping. (Boolean)

-trustedTargetRealms
Specifies a list of target realms to trust. Separate each realm name with the pipe character (|). (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureCSIOutbound('-securityDomainName testDomain -useServerIdentity true -messageAuthLevel
Supported')
```

- Using Jython list:

```
AdminTask.configureCSIOutbound(['securityDomainName', 'testDomain', '-useServerIdentity', 'true',
'-messageAuthLevel', 'Supported'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureCSIOutbound('-interactive')
```

getCSIInboundInfo

The getCSIInboundInfo command displays information about the Common Secure Interoperability (CSI) inbound settings for the global security configuration or for a security domain.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-displayModel
Specifies the output format of the configuration information. Specify `true` to return an attribute list of the model. Specify `false` to display an attribute of the value used to create the object. (Boolean)

Return value

The command returns an attribute list of the attributes and values of the CSI inbound object.

Batch mode example usage

- Using Jython string:

```
AdminTask.getCSIInboundInfo('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getCSIInboundInfo(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getCSIInboundInfo('-interactive')
```

getCSIOutboundInfo

The getCSIOutboundInfo command displays information for the CSI outbound settings for the global security configuration or for a security domain.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

-displayModel
Specifies the output format of the configuration information. Specify `true` to return an attribute list of the model. Specify `false` to display an attribute of the value used to create the object. (Boolean)

Return value

The command returns an attribute list that contains the attributes and values of the CSI outbound configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.getCSIOutboundInfo('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getCSIOutboundInfo(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getCSIOutboundInfo('-interactive')
```

unconfigureCSIInbound

The unconfigureCSIInbound command removes the CSI inbound information from a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureCSIInbound('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureCSIInbound(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureCSIInbound('-interactive')
```

unconfigureCSIOutbound

The unconfigureCSIOutbound command removes the CSI outbound information from a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureCSIOutbound('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureCSIOutbound(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureCSIOutbound('-interactive')
```

configureInterceptor

The configureInterceptor command modifies an existing interceptor or creates an interceptor if one does not exist.

Target object

None.

Required parameters

-interceptor
Specifies the trust association interceptor class name. (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain. If you do not specify a security domain, the command assigns the global security configuration. (String)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureInterceptor('-interceptor com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus
-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.configureInterceptor(['interceptor', 'com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus',
'-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureInterceptor('-interactive')
```

configureTrustAssociation

The configureTrustAssociation command enables or disable the trust association. If the security domain does not have a trust association defined, the application server copies each trust association and its interceptors from the global security configuration.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. (String)

-enable
Specifies whether to enable trust association to act as a reverse proxy server. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureTrustAssociation('-securityDomainName testDomain -enable true')
```

- Using Jython list:

```
AdminTask.configureTrustAssociation(['securityDomainName', 'testDomain', '-enable', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureTrustAssociation('-interactive')
```

getTrustAssociationInfo

The getTrustAssociationInfo command displays configuration information for trust association.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an attribute list that contains attributes and values for trust association.

Batch mode example usage

- Using Jython string:

```
AdminTask.getTrustAssociationInfo('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getTrustAssociationInfo(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getTrustAssociationInfo('-interactive')
```

listInterceptors

The listInterceptors command displays the trust association interceptors that are configured in the global security or security domain configuration.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an array list of each interceptor and the associated custom properties.

Batch mode example usage

- Using Jython string:

```
AdminTask.listInterceptors('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.listInterceptors(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listInterceptors('-interactive')
```

unconfigureInterceptor

The unconfigureInterceptor command removes a trust association interceptor from the global security configuration or from a security domain.

Target object

None.

Required parameters

-interceptor
Specifies the trust association interceptor class name. (String)

Optional parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureInterceptor('-interceptor com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus
-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureInterceptor(['interceptor', 'com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus',
'-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureInterceptor('-interactive')
```

unconfigureTrustAssociation

The unconfigureTrustAssociation command removes the trust association object from a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureTrustAssociation('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureTrustAssociation(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureTrustAssociation('
```

applyWizardSettings

The applyWizardSettings command can be used to automate the global security configuration.

Target object

None.

Optional parameters

-secureApps
Specifies to secure applications.

-secureLocalResources
Specifies to secure local resources such as data sets and MVS™ commands.

-userRegistryType
Specifies whether the user is a user, a group, or a group member.

-ldapServerType
Specifies the type of LDAP server that is being used. The default value is IDS51.

-ldapHostName
Specifies the LDAP host name.

-ldapPort
Specifies the LDAP port name.

-ldapBaseDN
Specifies the LDAP base dynamic member attribute.

-ldapBindDN
Dynamically updates LDAP binding information.

-ldapBindPassword
Dynamically updates LDAP binding password information.

-adminName
Refers to the name of an administrator account on the remote target machine.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.applyWizardSettings('[-secureApps true -secureLocalResources false
-userRegistryType LDAPUserRegistry -ldapServerType IBM_DIRECTORY_SERVER
-ldapHostName '+ldapServer+' -ldapPort 389 -ldapBaseDN o=ibm,c=us -ldapBindDN
cn=root -ldapBindPassword a1x4meok -adminName '+adminUsername+' ]'))
```

configureAuthzConfig

The configureAuthzConfig command configures an external Java Authorization Contract for Containers (JACC) authorization provider in a security domain or the global security configuration.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security configuration. (String)

-useJACCProvider
Specifies whether to use a JACC provider. Specify `true` to use a JACC provider. (Boolean)

-name
Specifies the name of the JACC provider to use. (String)

-description
Specifies a description of the JACC provider. (String)

-j2eePolicyImplClassName
Specifies the class name of an implementation class that represents the javax.security.jacc.policy.provider property according to the specification. (String)

-policyConfigurationFactoryImplClassName
Specifies the class name of an implementation class that represents the javax.security.jacc.PolicyConfigurationFactory.provider property. (String)

-roleConfigurationFactoryImplClassName
Specifies the class name of an implementation class that implements the com.ibm.wsspi.security.authorization.RoleConfigurationFactory interface. (String)

-requiresEJBArgumentsPolicyContextHandler
Specifies whether policy providers require the Enterprise JavaBeans arguments policy context handler to make access decisions. Specify `true` to enable this option. (Boolean)

-initializeJACCProviderClassName
Specifies the class name of an implementation class that implements the com.ibm.wsspi.security.authorization.IntializeJACCProvider interface.(String)

-supportsDynamicModuleUpdates
Specifies whether the provider supports dynamic changes to the web modules. Specify `true` to enable this option. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureAuthzConfig('[-securityDomainName testDomain -useJACCProvider true -name
testProvider -description "JACC provider for testing"]')
```

- Using Jython list:

```
AdminTask.configureAuthzConfig(['securityDomainName', 'testDomain', '-useJACCProvider', 'true', '-name',
'testProvider', '-description', 'JACC provider for testing'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureAuthzConfig('-interactive')
```

configureSingleSignon

The configureSingleSignon command configures a single sign-on object in global security.

Target object

None.

Optional parameters

-enable
Specifies whether to enable single sign-on. Specify `true` to enable single sign-on, or `false` to disable single sign-on. (Boolean)

-requiresSSL
Specifies whether single sign-on requests send through HTTPS. Specify `true` to enable this option. (Boolean)

-domainName
Specifies the domain name that contains a set of hosts to which the single sign-on applies. (String)

-interoperable
Specifies interoperability options. Specify `true` to send an interoperable cookie to the browser to support back-level servers. Specify `false` disable the sending of interoperable cookies. (Boolean)

-attributePropagation
Specifies whether to enable inbound security attribute propagation. Specify `true` to enable web inbound security attribution propagation. Specify `false` to use the single sign-on token to log in and recreate the Subject from the user registry. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureSingleSignon('-enable true -domainName mycompany.com')
```

- Using Jython list:

```
AdminTask.configureSingleSignon(['enable', 'true', '-domainName', 'mycompany.com'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.configureSingleSignon('-interactive')
```

getActiveSecuritySettings

The getActiveSecuritySettings command displays the active security settings for global security or a specific security domain.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security domain configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns the active security settings for the security domain of interest or the global security configuration, which includes the following settings:

- cacheTimeout
- issuePermissionWarning
- activeAuthMechanism
- enforceJava2Security
- appSecurityEnabled
- enableGlobalSecurity (global security only)
- adminPreferredAuthMech (global security only)
- activeAuthMechanism (global security only)
- activeUserRegistry
- enforceFineGrainedJCASecurity
- dynUpdateSSLConfig (global security only)
- useDomainQualifiedUserNames
- customProperties

Batch mode example usage

- Using Jython string:

```
AdminTask.getActiveSecuritySettings('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getActiveSecuritySettings(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getActiveSecuritySettings('-interactive')
```

getAuthzConfigInfo

The getAuthzConfigInfo command displays information about an external JACC authorization provider in a security domain or the global security configuration.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security domain configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command returns an attribute list that contains the attributes and values that are associated with the JACC authorization provider.

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuthzConfigInfo('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.getAuthzConfigInfo(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuthzConfigInfo('-interactive')
```

getSingleSignon

The getSingleSignon command displays configuration information about the single sign-on object as defined in the global security configuration.

Target object

None.

Optional parameters

None.

Return value

The command returns an attribute list that contains the attributes and values of the single sign-on configuration.

Batch mode example usage

- Using Jython:

```
AdminTask.getSingleSignon()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getSingleSignon('-interactive')
```

setAdminActiveSecuritySettings

The setAdminActiveSecuritySettings command sets the active security settings on the global security object.

Note: To set the security settings for a security domain, see the setAppActiveSecuritySettings command.

Target object

None.

Optional parameters

-enableGlobalSecurity
Specifies whether to enable global security. Specify `true` to enable global security, or specify `false` to disable global security. (Boolean)

-cacheTimeout
Specifies the amount of time, in seconds, before authentication data becomes invalid. (Integer)

-issuePermissionWarning
Specifies whether to issue a warning during application installation if the application requires security permissions. Specify `true` to enable the warning notification, or specify `false` to disable the warning notification. (Boolean)

-enforceJava2Security
Specifies whether to enable Java Platform, Enterprise Edition (Java EE) security. Specify `true` to enable Java EE security permissions checking, or specify `false` to disable Java EE security. (Boolean)

-enforceFineGrainedJCASecurity
Specifies whether to restrict application access. Specify `true` to restrict application access to sensitive Java EE Connector Architecture (JCA) mapping authentication data. (Boolean)

-appSecurityEnabled
Specifies whether to enable application-level security. Specify `true` to enable application level security, or specify `false` to disable application-level security. (Boolean)

-dynUpdateSSLConfig
Specifies whether to dynamically update SSL configuration changes. Specify `true` to update SSL configuration changes dynamically, or specify `false` to update the SSL configuration when the server starts. (Boolean)

-activeAuthMechanism
Specifies the active authentication mechanism. Specify `LTPA` for LTPA authentication, `KRB5` for Kerberos authentication, or `RSAToken` for RSA token authorization. (String)

-adminPreferredAuthMech
Specifies the preferred authentication mechanism. Specify `LTPA` for LTPA authentication, `KRB5` for Kerberos authentication, or `RSAToken` for RSA token authorization. (String)

-activeUserRegistry
Specifies the active user registry for the server. (String)

Specify one of the following values:

CustomUserRegistry
This option enables you to specify a custom user registry as the active user registry for the server.

LDAPUserRegistry
This option enables you to specify an LDAP user registry as the active user registry for the server.

LocalOSUserRegistry
This option enables you to specify the local operating system user registry as the active user registry for the server.

WIMUserRegistry
This option enables you to specify a federated repository as the active user registry for the server.

-useDomainQualifiedUserNames
Specifies the type of user name to use. Specify `true` to use domain qualified user names, or specify `false` to use the short name. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAdminActiveSecuritySettings('-enableGlobalSecurity true -cacheTimeout 300
-enforceJava2Security true -appSecurityEnabled true -activeUserRegistry LDAPUserRegistry')
```

- Using Jython list:

```
AdminTask.setAdminActiveSecuritySettings(['enableGlobalSecurity', 'true', '-cacheTimeout',
'300', '-enforceJava2Security', 'true', '-appSecurityEnabled', 'true' '-activeUserRegistry',
'LDAPUserRegistry'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setAdminActiveSecuritySettings('-interactive')
```

setAppActiveSecuritySettings

The setAppActiveSecuritySettings command sets the active security settings on a security domain.

Note: To set the security settings for global security, see the setAdminActiveSecuritySettings command.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Optional parameters

-cacheTimeout
Specifies the amount of time, in seconds, before authentication data becomes invalid. (Integer)

-issuePermissionWarning
Specifies whether to issue a warning during application installation if the application requires security permissions. Specify `true` to enable the warning notification, or specify `false` to disable the warning notification. (Boolean)

-enforceJava2Security
Specifies whether to enable Java Platform, Enterprise Edition (Java EE) security. Specify `true` to enable Java EE security permissions checking, or specify `false` to disable Java EE security. (Boolean)

-enforceFineGrainedJCASecurity
Specifies whether to restrict application access. Specify `true` to restrict application access to sensitive Java EE Connector Architecture (JCA) mapping authentication data. (Boolean)

-appSecurityEnabled
Specifies whether to enable application-level security. Specify `true` to enable application level security, or specify `false` to disable application-level security. (Boolean)

-activeUserRegistry
Specifies the active user registry for the server. (String)

-useDomainQualifiedUserNames
Specifies the type of user name to use. Specify `true` to use domain qualified user names, or specify `false` to use the short name. (Boolean)

-customProperties
Specifies a comma separated list of quoted attribute and value pairs that the system stores as custom properties on the user registry object. For example, use the format: `"attr1=value1","attr2=value2"` (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAppActiveSecuritySettings('-securityDomainName testDomain -issuePermissionWarning false
-enforceFineGrainedJCASecurity true')
```

- Using Jython list:

```
AdminTask.setAppActiveSecuritySettings(['securityDomainName', 'testDomain', '-issuePermissionWarning',
'false', '-enforceFineGrainedJCASecurity', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setAppActiveSecuritySettings('-interactive')
```

unconfigureAuthzConfig

The unconfigureAuthzConfig command removes an external JACC authorization provider from the global security configuration or a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureAuthzConfig('-securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureAuthzConfig(['securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unconfigureAuthzConfig('-interactive')
```

unsetAppActiveSecuritySettings

The unsetAppActiveSecuritySettings command removes an attribute from the global security configuration or a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security configuration. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter. (String)

Optional parameters

-unsetAppSecurityEnabled
Specifies whether to remove the attribute that enables application security. Specify `true` to remove the attribute. (Boolean)

-unsetActiveUserRegistry
Specifies whether to remove the active user registry attribute. Specify `true` to remove the attribute. (Boolean)

-unsetUseDomainQualifiedUserNames
Specifies whether to remove the user domain qualified user names attribute. Specify `true` to remove the attribute. (Boolean)

-unsetEnforceJava2Security
Specifies whether to remove the Java EE security attribute. Specify `true` to remove the attribute. (Boolean)

-unsetEnforceFineGrainedJCASecurity
Specifies whether to remove the fine-grained JCA security attribute. Specify `true` to remove the attribute. (Boolean)

-unsetIssuePermissionWarning
Specifies whether to remove the attribute that issues user permission warnings. Specify `true` to remove the attribute. (Boolean)

-unsetCacheTimeout
Specifies whether to remove the cache timeout attribute. Specify `true` to remove the attribute. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unsetAppActiveSecuritySettings('-securityDomainName testDomain -unsetAppSecurityEnabled true
-unsetPermissionWarning true')
```

- Using Jython list:

```
AdminTask.unsetAppActiveSecuritySettings(['securityDomainName', 'testDomain', '-unsetAppSecurityEnabled',
'true', '-unsetPermissionWarning', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unsetAppActiveSecuritySettings('-interactive')
```

SecurityRealmInfoCommands command group for the AdminTask object

You can use the Jython scripting language to manage security realm configurations with the wsadmin tool. Use the commands and parameters in the SecurityRealmInfoCommands group to query and manage trusted realms.

Use the following commands to manage trusted realms in your security configuration:

addTrustedRealms

The addTrustedRealms command adds a realm or list of realms to the list of trusted realms for global security or in a security domain.

Target object

None.

Required parameters

-communicationType
Specifies whether to trusted realms to inbound or outbound communication. Specify `inbound` to configure inbound communication. Specify `outbound` to configure outbound communication. (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain of interest. If you do not specify a value for this parameter, the command uses the global security configuration. (String)

-realmList
Specifies a realm or list of realms to configure as trusted realms. (String)

Separate each realm in the list with the pipe character (|) as the following example demonstrates: `realm1|realm2|realm3`

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.addTrustedRealms('-communicationType inbound -securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.addTrustedRealms(['-communicationType', 'inbound', '-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addTrustedRealms('-interactive')
```

configureTrustedRealms

The configureTrustedRealms command configures trusted realms. Use this command to replace the list of trusted realms and to clear each realm from the list. To add realms to the trusted realm list, use the addInboundTrustedRealm command.

Target object

None.

Required parameters

-communicationType
Specifies whether to configure the security domains, realms, or global security configuration for inbound or outbound communication. Specify `inbound` to configure inbound communication. Specify `outbound` to configure outbound communication. (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain of interest. If you do not specify a value for this parameter, the command uses the global security configuration. (String)

-realmList
Specifies a list of realms to configure as trusted realms. (String)

Separate each realm in the list with the pipe character (|) as the following example demonstrates: `realm1|realm2|realm3`

-trustAllRealms
Specifies whether to trust all realms. Specify `true` to trust all realms. If you specify `true` for this parameter, the command does not use the -realmList parameter. (Boolean)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.configureTrustedRealms('-communicationType inbound -realmList realm1|realm2|realm3')
```

- Using Jython list:

```
AdminTask.configureTrustedRealms(['-communicationType', 'inbound', '-realmList', 'realm1|realm2|realm3'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.configureTrustedRealms('-interactive')
```

- Using Jython list:

listRegistryGroups

The listRegistryGroups command displays the groups in the user registry that belong to the security realm, security domain, or resource name of interest.

Target object

None.

Optional parameters

-securityRealmName
Specifies name of the security realm of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters. (String)

-resourceName
Specifies the name of the resource of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters. (String)

-securityDomainName
Specifies the name of the security domain of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters.(String)

-displayAccessIds
Specifies whether to display the access IDs for each group. Specify `true` to display the access ID and group name for each group that the command returns. (Boolean)

-groupFilter
Specifies a filter that the command uses to query for groups. For example, specify `test*` to return groups that begin with the `test` string. By default, the command returns all groups. (String)

-numberOfGroups
Specifies the number of groups to return. The default number of groups that the command displays is `20`. (Integer)

Return value

The command returns an array of group names. If you specified the -displayAccessId parameter, the command returns an array of attribute lists which contain the group name and group access ID.

Batch mode example usage

- Using Jython string:

```
AdminTask.listRegistryGroups('-securityDomainName myTestDomain -groupFilter test* -numberOfGroups 10')
```

- Using Jython list:

```
AdminTask.listRegistryGroups(['-securityDomainName', 'myTestDomain', '-groupFilter', 'test*', '-numberOfGroups', '10'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listRegistryGroups('-interactive')
```

listRegistryUsers

The listRegistryUsers command displays the users in the user registry for a specific security realm, resource name, or domain name.

Target object

None.

Optional parameters

-securityDomainName
Specifies the name of the security domain of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters. If you do not specify the securityDomainName, resourceName, or securityRealmName parameter, the system uses the active user registry from the global security configuration. (String)

-resourceName
Specifies the name of the resource of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters. If you do not specify the securityDomainName, resourceName, or securityRealmName parameter, the system uses the active user registry from the global security configuration. (String)

-securityRealmName
Specifies the name of the security realm of interest. The securityDomainName, resourceName, and securityRealmName parameters are mutually exclusive. Do not specify more than one of these parameters. If you do not specify the securityDomainName, resourceName, or securityRealmName parameter, the system uses the active user registry from the global security configuration. (String)

-displayAccessIds
Specifies whether to display the access IDs for each group. Specify `true` to display the access ID and group name for each group that the command returns. (Boolean)

-userFilter
Specifies the filter that the command uses to query for users. For example, specify `test*` to display each user name that starts with the `test` string. By default, the command returns all users. (String)

-numberOfUsers
Specifies the number of users to return. The default number of groups that the command displays is `20`. (Integer)

Return value

The command returns an array of user names. If you specify the -displayAccessId parameter, the command returns an array of attribute lists that contain the user ID and user access IDs.

Batch mode example usage

- Using Jython string:

```
AdminTask.listRegistryUsers('-securityRealmName defaultWIMFileBasedRealm  -displayAccessIds true')
```

- Using Jython list:

```
AdminTask.listRegistryUsers(['-securityRealmName', 'defaultWIMFileBasedRealm', '-displayAccessIds', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listRegistryUsers('-interactive')
```

listSecurityRealms

The listSecurityRealms command displays each security realm from global security configuration and the security domains.

Target object

None.

Return value

The command returns an array of realm names.

Batch mode example usage

- Using Jython string:

```
AdminTask.listSecurityRealms()
```

- Using Jython list:

```
AdminTask.listSecurityRealms()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSecurityRealms('-interactive')
```

listTrustedRealms

The listTrustedRealms command displays a list of trusted realms for a security domain, resource, or realm. If you do not specify a security domain, resource name, or realm name, then the command returns a list of trusted realms from the global security configuration. The securityRealmName, resourceName, and securityDomainName parameters are mutually exclusive.

Target object

None.

Required parameters

-communicationType
Specifies whether to list the trusted realms for inbound or outbound communication. Specify `inbound` to configure inbound communication. Specify `outbound` to configure outbound communication. (String)

Optional parameters

-securityRealmName
Specifies name of the security realm of interest. If you use this parameter, do not use the resourceName or securityDomainName parameters. (String)

-resourceName
Specifies the name of the resource of interest. If you use this parameter, do not use the securityRealmName or securityDomainName parameters. (String)

-securityDomainName
Specifies the name of the security domain of interest. If you use this parameter, do not use the resourceName or securityRealmName parameters. (String)

-expandRealmList
Specifies whether to return each realm name when the trustAllRealms property is enabled. Specify `true` to return each realm name. Specify `false` to return the trustAllRealms property. (Boolean)

-includeCurrentRealm
Specifies whether to include the current realm in the list of trusted realms. Specify `true` to include the current realm, or specify `false` to exclude the current realm from the list of trusted realms. (Boolean)

Return value

The command returns an array of trusted realm names. If the realm, resource, or security domain of interest is configured to trust all realms, the command returns the `trustAllRealms` string.

Batch mode example usage

- Using Jython string:

```
AdminTask.listTrustedRealms('-communicationType inbound -resourceName myApplication')
```

- Using Jython list:

```
AdminTask.listTrustedRealms(['-communicationType', 'inbound', '-resourceName', 'myApplication'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listTrustedRealms('-interactive')
```

removeTrustedRealms

The removeTrustedRealms command removes realms from a trusted realm list in a security domain or in the global security configuration.

Target object

None.

Required parameters

-communicationType
Specifies whether to remove trusted realms from inbound or outbound communication. Specify `inbound` to configure inbound communication. Specify `outbound` to configure outbound communication. (String)

-realmList
Specifies a list of realms to remove from trusted realms. (String)

Separate each realm in the list with the pipe character (|) as the following example demonstrates: `realm1|realm2|realm3`

Optional parameters

-securityDomainName
Specifies the name of the security domain of interest. If you do not specify a security domain, the command uses the global security configuration. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeTrustedRealms('-communicationType inbound -realmList realm1|realm2|realm3')
```

- Using Jython list:

```
AdminTask.removeTrustedRealms(['-communicationType inbound -realmList realm1|realm2|realm3'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeTrustedRealms('-interactive')
```

unconfigureTrustedRealms

The unconfigureTrustedRealms command removes the trusted realm object from the configuration.

Target object

None.

Required parameters

-communicationType
Specifies whether to unconfigure the trusted realms for inbound or outbound communication. Specify `inbound` to remove inbound communication configurations. Specify `outbound` to remove outbound communication configurations. (String)

Optional parameters

-securityDomainName
Specifies the name of the security domain of interest. If you do not specify a security domain, the command uses the global security configuration. (String)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.unconfigureTrustedRealms('-communicationType inbound -securityDomainName testDomain')
```

- Using Jython list:

```
AdminTask.unconfigureTrustedRealms(['-communicationType', 'inbound', '-securityDomainName', 'testDomain'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.unconfigureTrustedRealms('-interactive')
```

NamingAuthzCommands command group for the AdminTask object

You can use the Jython scripting language to configure naming roles for groups and users with the wsadmin tool. Use the commands and parameters in the NamingAuthzCommands group to assign, remove, and query naming role configuration. CosNaming security offers increased granularity of security control over CosNaming functions.

A number of naming roles are defined to provide the degrees of authority that are needed to perform certain application server naming service functions. The authorization policy is only enforced when global security is enabled.

Use the following commands to manage the naming service functions:

- listGroupsForNamingRoles
- listUsersForNamingRoles
- mapGroupsToNamingRole
- mapUsersToNamingRole
- removeGroupsFromNamingRole
- removeUsersFromNamingRole

listGroupsForNamingRoles

The listGroupsForNamingRoles command displays the groups and special subjects that are mapped to the naming roles.

Target object

None.

Return value

The command returns a list of the groups and special subjects associated with each naming role.

Batch mode example usage

- Using Jython:

```
AdminTask.listGroupsForNamingRoles()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listGroupsForNamingRoles('-interactive')
```

listUsersForNamingRoles

The listUsersForNamingRoles command displays the users that are mapped to the naming roles.

Target object

None.

Return value

The command returns a list of the users associated with each naming role.

Batch mode example usage

- Using Jython:

```
AdminTask.listUsersForNamingRoles()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listUsersForNamingRoles('-interactive')
```

mapGroupsToNamingRole

The mapGroupsToNamingRole command maps groups, special subjects, or groups and special subjects to the naming roles.

Target object

None.

Required parameters

-roleName
Specifies the name of the naming role. (String)

Table 10. Name space security roles. Four name space security roles are available: `CosNamingRead`, `CosNamingWrite`, `CosNamingCreate`, and `CosNamingDelete`. The roles have authority levels from low to high, as the following table defines:

Role name	Description
`CosNamingRead`	You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The `EVERYONE` special-subject is the default policy for this role.
`CosNamingWrite`	You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations.
`CosNamingCreate`	You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.
`CosNamingDelete`	You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Optional parameters

-groupids
Specifies the names of the groups to map to the naming roles. (String[])

-accessids
Specifies the access IDs of the users in the format `<group:realmName/uniqueID>`. (String[])

-specialSubjects
Specifies the special subjects to map. (String[])

Table 11. Special subjects. The special subjects include EVERYONE, ALLAUTHENTICATED, ALLAUTHENTICATEDINTRUSTEDREALMS, as the following table defines:

Header	Header
EVERYONE	Maps everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.
ALLAUTHENTICATED	Maps each authenticated user to a specified role. When you map each authenticated user to a specified role, each valid user in the current registry who has been authenticated can access resources that are protected by this role.
ALLAUTHENTICATEDINTRUSTEDREALMS	Maps each authenticated user to a specified role. When you map each authenticated user to a specified role, each valid user in the current registry who has been authenticated can access resources that are protected by this role in the trusted realm.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.mapGroupsToNamingRole('[-roleName CosNamingCreate -groupids [group1, group2]]')
```

- Using Jython list:

```
AdminTask.mapGroupsToNamingRole(['-roleName', 'CosNamingCreate', '-groupids', '[group1, group2]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.mapGroupsToNamingRole('-interactive')
```

mapUsersToNamingRole

The mapUsersToNamingRole command maps users to the naming roles.

Target object

None.

Required parameters

-roleName
Specifies the name of the naming role. (String)

Table 12. Name space security roles. Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The roles have authority levels from low to high, as the following table defines:

Role name	Description
CosNamingRead	You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.
CosNamingWrite	You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations.
CosNamingCreate	You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.
CosNamingDelete	You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Optional parameters

-userids
Specifies the user IDs to map to the naming roles of interest. (String[])

-accessids
Specifies the access IDs of the users in the format <user:realmName/uniqueID>. (String[])

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.mapUsersToNamingRole([-roleName CosNamingDelete -userids [user1, user2, user3]]')
```

- Using Jython list:

```
AdminTask.mapUsersToNamingRole(['-roleName', 'CosNamingDelete', '-userids', '[user1, user2, user3]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.mapUsersToNamingRole('-interactive']
```

removeGroupsFromNamingRole

The removeGroupsFromNamingRole command removes groups, special subjects, or groups and special subjects from a naming role.

Target object

None.

Required parameters

-roleName
Specifies the name of the naming role. (String)

Table 13. Name space security roles. Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The roles have authority levels from low to high, as the following table defines:

Role name	Description
CosNamingRead	You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.
CosNamingWrite	You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations.
CosNamingCreate	You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.
CosNamingDelete	You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Optional parameters

-groupids
Specifies the names of the groups to remove from the naming roles of interest. (String[])

-specialSubjects
Specifies the special subjects to remove. (String[])

Table 14. Special subjects. The special subjects include EVERYONE, ALLAUTHENTICATED, ALLAUTHENTICATEDINTRUSTEDREALMS, as the following table defines:

Header	Header
EVERYONE	Maps everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.
ALLAUTHENTICATED	Maps each authenticated user to a specified role. When you map each authenticated user to a specified role, each valid user in the current registry who has been authenticated can access resources that are protected by this role.
ALLAUTHENTICATEDINTRUSTEDREALMS	Maps each authenticated user to a specified role. When you map each authenticated user to a specified role, each valid user in the current registry who has been authenticated can access resources that are protected by this role in the trusted realm.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeGroupsFromNamingRole('-roleName CosNamingRead -groupids [group1, group2] -specialSubjects EVERYONE')
```

- Using Jython list:

```
AdminTask.removeGroupsFromNamingRole(['-roleName', 'CosNamingRead', '-groupids', '[group1, group2]', '-specialSubjects', 'EVERYONE'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeGroupsFromNamingRole('-interactive')
```

removeUsersFromNamingRole

The removeUsersFromNamingRole command removes users from a naming role.

Target object

None.

Required parameters

-roleName
Specifies the name of the naming role. (String)

Table 15. Name space security roles. Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The roles have authority levels from low to high, as the following table defines:

Role name	Description
CosNamingRead	You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.
CosNamingWrite	You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations.
CosNamingCreate	You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.
CosNamingDelete	You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Optional parameters

-userids
Specifies the user IDs to remove from the naming roles of interest. (String[])

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeUsersFromNamingRole('-roleName CosNamingRead')
```

- Using Jython list:

```
AdminTask.removeUsersFromNamingRole(['-roleName', 'CosNamingRead'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeUsersFromNamingRole('-interactive')
```

Utility scripts

The scripting library provides multiple script procedures to automate your application configurations. This topic provides usage information for scripts that set notification options, save configuration changes, and display scripting library information.

Each utility script procedure is located in the *app_server_root*/scriptLibraries/utilities/V70 directory. Use the following script procedures to perform utility functions:

convertToList

This script converts a string to a list. For example, the AdminApp.list() command returns a string of application names. Use the convertToList script to change the output to a list format, such as ['DefaultApplication', 'a1', 'a2', 'ivtApp', 'query'].

Table 16. convertToList argument description. Run the script to return a string output and set the output to a variable.

Argument	Description
variable	Specifies the name of the variable that contains the string to convert to a list.

Syntax

```
AdminUtilities.convertToList(variable)
```

Example usage

```
apps=AdminApp.list()
AdminUtilities.convertToList(apps)
```

debugNotice

This script sets the debug notice text.

Table 17. debugNotice argument description. Run the script to specify the message argument.

Argument	Description
message	Specifies the message text for the debug notice.

Syntax

```
AdminUtilities.debugNotice(message)
```

Example usage

```
AdminUtilities.debugNotice("Server is started")
```

getExceptionText

This script displays the exception message for a specific exception type, exception value, or traceback information.

Table 18. getExceptionText argument descriptions. Run the script to specify the type, value, or traceback arguments.

Argument	Description
type	Specifies the exception type of interest. The exception type represents the class object of the exception.
value	Specifies the exception value of interest. The value represents the instance object that is the argument of the exception or the second argument of the raise statement.
traceback	Specifies the traceback information of interest. The traceback object contains special attributes, including the line number where the error occurred. Do not assign *traceback* to a local variable in the function that handles the exception, as this assignment creates a circular reference.

Syntax

```
AdminUtilities.getExceptionText(type, value, traceback)
```

Example usage

```
AdminUtilities.getExceptionText("com.ibm.ws.scripting.ScriptingException
", "com.ibm.ws.scripting.ScriptingException: AdminControl service not available"
, "")
```

fail

This script sets the failure message.

Table 19. fail argument description. Run the script to specify the message argument.

Argument	Description
message	Specifies the message text for the failure notice.

Syntax

```
AdminUtilities.fail(message)
```

Example usage

```
AdminUtilities.fail("The script failed")
```

fileSearch

This script searches the file system based on a specific path or directory.

Table 20. fileSearch argument descriptions. Run the script to specify the path or directory arguments.

Argument	Description
path	Specifies the file path to search for a specific file.
directory	Specifies the directory to search for a specific file.

Syntax

```
AdminUtilities.fileSearch(path, directory)
```

Example usage

```
Paths = []
Directory = java.io.File("//WebSphere//AppServer//scriptLibraries")
AdminUtilities.fileSearch(directory, paths)
```

getResourceBundle

This script displays an instance for the resource bundle of interest.

Table 21. getResourceBundle argument description. Run the script to specify the bundle name argument.

Argument	Description
bundleName	Specifies the name of the bundle of interest. For example, to get a message object from the ScriptingLibraryMessage resource bundle, specify `com.ibm.ws.scripting.resources.scriptLibraryMessage`.

Syntax

```
AdminUtilities.getResourceBundle(bundleName)
```

Example usage

```
AdminUtilities.getResourceBundle("com.ibm.ws.scripting.resources.scriptLibraryMessage")
```

getScriptLibraryFiles

This script displays the file path and file names for each script library file.

Syntax

```
AdminUtilities.getScriptLibraryFiles()
```

Example usage

```
AdminUtilities.getScriptLibraryFiles()
```

getScriptLibraryList

This script displays each script name in the script library.

Syntax

```
AdminUtilities.getScriptLibraryList()
```

Example usage

```
AdminUtilities.getScriptLibraryList()
```

getScriptLibraryPath

This script displays the file path to get to the script library files on your file system.

Syntax

```
AdminUtilities.getScriptLibraryPath()
```

Example usage

```
AdminUtilities.getScriptLibraryPath()
```

help

This script displays help information for the AdminUtilities script library, including general library information, script names, and script descriptions.

Table 22. help argument description. Run the script to obtain information about the script of interest.

Argument	Description
scriptName	Optionally specifies the name of the AdminUtilities script of interest.

Syntax

```
AdminUtilities.help(scriptName)
```

Example usage

```
AdminUtilities.help("sleepDelay")
```

infoNotice

This script sets the text for the information notice of a command or script.

Table 23. infoNotice argument description. Run the script to specify the message argument.

Argument	Description
message	Specifies the message text or a message ID such as `"Application is installed"` or `resourceBundle.getString("WASX7115I")`.

Syntax

```
AdminUtilities.infoNotice(message)
```

Example usage

```
AdminUtilities.infoNotice(resourceBundle.getString("WASX7115I"))
```

save

This script saves the configuration changes to your system.

Syntax

```
AdminUtilities.save()
```

Example usage

```
AdminUtilities.save()
```

setDebugNotices

This script enables and disables debug notices.

Table 24. setDebugNotices argument description. Run the script to specify the debug argument.

Argument	Description
debug	Specifies whether to enable or disable debug notices. Specify `true` to enable debug notices, or `false` to disable debug notices.

Syntax

```
AdminUtilities.setDebugNotices(debug)
```

Example usage

```
AdminUtilities.setDebugNotices("true")
```

setFailOnErrorDefault

This script enables or disables the fail on error behavior.

Table 25. setFailOnErrorDefault argument description. Run the script to specify the fail on error argument.

Argument	Description
failOnError	Specifies whether to enable or disable the fail on error behavior. Specify `true` to enable the fail on error behavior, or `false` to disable the behavior.

Syntax

```
AdminUtilities.setFailOnErrorDefault(failOnError)
```

Example usage

```
AdminUtilities.setFailOnErrorDefault("false")
```

sleepDelay

This script sets the number of seconds that the system waits for completion during two operations.

Table 26. sleepDelay argument description. Run the script to specify the delay seconds argument.

Argument	Description
delaySeconds	Specifies the number of seconds to wait for completion.

Syntax

```
AdminUtilities.sleepDelay(delaySeconds)
```

Example usage

```
AdminUtilities.sleepDelay("10")
```

warningNotice

This script sets the text to display as the warning message.

Table 27. warningNotice argument description. Run the script to specify the message argument.

Argument	Description
message	Specifies the non-translated text for the warning notice or a message ID such as `resourceBundle.getString("WASX7411W")`.

Syntax

AdminUtilities.warningNotice(*message*)

Example usage

AdminUtilities.warningNotice(*resourceBundle.getString("WASX7411W")*)

configureAutoSave

This script enables and disables the automatic saving of configuration changes to the master configuration repository.

Table 28. configureAutoSave argument description. Run the script to specify the autosave argument.

Argument	Description
autosave	Specifies whether to save configuration changes to the master configuration repository automatically. The default value is `true`.

Syntax

AdminUtilities.configureAutoSave(*autosave*)

Example usage

AdminUtilities.configureAutoSave(*false*)

Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility

You can use the wsadmin utility to configure Tivoli Access Manager security for WebSphere Application Server.

About this task

Verify that all the managed servers, including node agents, are started. The following configuration is performed once on the deployment manager server. The configuration parameters are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers require their own restart for the configuration changes to take effect.

Procedure

1. Start WebSphere Application Server.
2. At the **wsadmin** prompt, enter the following command:

```
$AdminTask configureTAM -interactive
```

Table 29. Commands for configuring, reconfiguring, and unconfiguring Tivoli Access Manager. The following table lists the information that you are asked to provide for the configureTAM command. The table also lists the properties that apply to the unconfigureTAM and reconfigureTAM commands.

Property	Default	Relevant command	Description
Websphere Application Server node name	*	• configureTAM • reconfigureTAM • unconfigureTAM	Specify a single node or enter an asterisk (*) to run the configuration task on all of the application server instances including the deployment manager, node agents, and servers.
Tivoli Access Manager Policy Server	Default port: 7135	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration.
Tivoli Access Manager Authorization Server	Default port: 7136	• configureTAM • reconfigureTAM	Enter the name, port, and priority of each configured Tivoli Access Manager authorization server. Use the format *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration. You can specify more than one authorization server by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: auth_server1**:**7136**:**1,auth_server2**:**7137**:**2. A priority of 1 is still required when you use a single authorization server.

Table 29. Commands for configuring, reconfiguring, and unconfiguring Tivoli Access Manager (continued). The following table lists the information that you are asked to provide for the configureTAM command. The table also lists the properties that apply to the unconfigureTAM and reconfigureTAM commands.

Property	Default	Relevant command	Description
Websphere Application Server administrator's distinguished name		• configureTAM • reconfigureTAM	Enter the full distinguished name of the security primary administrator ID for WebSphere Application Server as created in the "Creating the security administrative user" topic in the *Securing applications and their environment* PDF. For example: cn=wasadmin,o=*organization*,c=*country*
Tivoli Access Manager user registry distinguished name suffix		• configureTAM • reconfigureTAM	Enter the suffix that you have set up in the user registry to contain the user and groups for Tivoli Access Manager. For example: o=*organization*,c=*country*
Tivoli Access Manager administrator's user name	sec_master	• configureTAM • reconfigureTAM • unconfigureTAM	Enter the Tivoli Access Manager administration user ID that you created when you configured Tivoli Access Manager. This ID is usually sec_master.
Tivoli Access Manager administrator's user password		• configureTAM • reconfigureTAM • unconfigureTAM	Enter the password that is associated with the Tivoli Access Manager administration user ID.
Tivoli Access Manager security domain	Default	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default.
Embedded Tivoli Access Manager listening port set	8900:8999	• configureTAM • reconfigureTAM	WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999.
Defer	No	• configureTAM • reconfigureTAM • unconfigureTAM	Set this option to *yes* if you want to defer the configuration of the management server until the next restart. Set the option to *no* if you want the configuration of the management server to occur immediately. Managed servers are configured on their next restart.
Force	No	• reconfigureTAM • unconfigureTAM	Set this value to *yes* if you want to ignore errors during the unconfiguration process and allow the entire process to complete. Set the value to *no* if you want errors to stop the unconfiguration process. This option is especially useful if the environment needs to be cleaned up and problems are occurring that do not allow the entire cleanup process to complete successfully.

3. When all information is entered, select **F** to save the configuration properties or **C** to cancel from the configuration process and discard entered information.

What to do next

Now enable the JACC provider for Tivoli Access Manager - see the Enabling the JACC provider for Tivoli Access Manager article for more information.

Securing communications using the wsadmin tool

The application server provides several methods to secure communication between a server and a client. Use this topic to configure Secure Sockets Layer (SSL), keystores, certificate authorities, key sets and groups, and certificates.

Procedure

- Configure secure communications using SSL.

  Use the SSLConfigCommands, SSLConfigGroupCommands, DynamicSSLConfigSelections and SSLTransport command groups for the AdminTask object, and complete the following tasks to create and administer SSL configurations:

  – Create an SSL configuration at the node scope using scripting.

  – Automate SSL configurations using scripting.

- Create a keystore configuration.

  Use the KeyStoreCommands command group for the AdminTask object, and complete the following tasks to create and administer keystore configurations.

  – Update default key store passwords using scripting

- Create a certificate authority (CA) client configuration.

A CA client object contains all of the configuration information necessary to connect to a third-party CA server. Use the CAClientCommands command group for the AdminTask object, and complete the following tasks to create and administer CA client objects in your configuration:

  – Configure CA clients using scripting
  – Administer CA clients using scripting

- Administer certificate configurations.

  Use the CertificateRequestCommands, PersonalCertificateCommands, and SignerCertificateCommands command groups for the AdminTask object, and complete the following tasks to administer personal certificates, CA certificates, and self-signed certificates:

  – Create self-signed certificates using scripting
  – Configure CA certificates using scripting

- Create key sets and key groups.

  Use the KeySetCommands, KeySetGroupCommands, and KeyReferenceCommands command groups for the AdminTask object to create and administer key set and group configurations.

Creating an SSL configuration at the node scope using scripting

A Secure Socket Layer (SSL) configuration references many other configuration objects. To help you make valid selections for the new SSL configuration before you create it, view information about existing configuration objects. Information about existing objects is also useful when you create a node scoped SSL configuration using the **createSSLConfig** command of the AdminTask object.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

gotcha: The security.xml file is restricted. Therefore, if you need to make changes to the security.xml file, verify that your user ID has administrator role authorization. If you are using a user ID with operator role authorization, you can perform a node synchronization, but any changes that you made to the security.xml file are not synchronized.

About this task

To use the information in this task effectively, familiarize yourself with the instructions in the Creating a Secure Sockets Layer configuration topic.

Perform the following task to create an Secure Socket Layer (SSL) configuration at the node scope:

Procedure

1. List the existing configuration objects. Perform any of the following:
   - List some of the configuration objects that you may need when you create a new SSL configuration.

     For example, you want to see which management scopes have already been defined. If the one you need does not exist you will need to create it.

     – Using Jacl:

```
$AdminTask listManagementScopes {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02}
```

     – Using Jython:

```
AdminTask.listManagementScopes ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02]')
```

     This shows an existing cell scope and existing node scope that you can use. If you want to create a different scope, use the **createManagementScope** command of the AdminTask object to define a different one. The valid scope parameters are cell, nodegroup, node, server, cluster, and endpoint. See the Central management of SSL configurations article for more information on scope limitations.
   - List the key stores that exist in the configuration including key stores and trust stores.

– Using Jacl:

```
$AdminTask listKeyStores -all true
```

– Using Jython:

```
AdminTask.listKeyStores('-all true')
```

Example output:

```
CellDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1)
CellDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_2)
CellLTPAKeys(cells/BIRKT40Cell02|security.xml#KeyStore_3)
```

The previous example only lists the key stores for the default management scope which is also known as the cell scope. To obtain key stores for other scopes, specify the `scopeName` parameter, for example:

– Using Jacl:

```
$AdminTask listKeyStores {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 }
```

– Using Jython:

```
$AdminTask listKeyStores ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02]')
```

Example output:

```
CellDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1)
CellDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_2)
CellLTPAKeys(cells/BIRKT40Cell02|security.xml#KeyStore_3)
NodeDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1134610924357)
NodeDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_1134610924377)
```

- List specific trust or key managers. Be sure to display the object name for the trust managers. You will need the object name for the SSL configuration because you can specify multiple trust manager instances.

  – Using Jacl:

```
$AdminTask listTrustManagers {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true }
```

  – Using Jython:

```
AdminTask.listTrustManagers ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true]')
```

  Example output:

```
IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1)
IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_2)
IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924357)
IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924377)
```

2. Create the node-scoped SSL configuration in interactive mode. Now that we have the information we need to choose from, we need to decide if these objects are sufficient or if we need to create new ones. For now, we will reuse what we've already got in the configuration and save creating new instances to task documents specific to those objects.

   - Using Jacl:

```
$AdminTask createSSLConfig -interactive
```

   - Using Jython:

```
AdminTask.createSSLConfig ('[-interactive]')
```

   Example output:

```
Create a SSL Configuration.

*SSL Configuration Alias (alias): BIRKT40Node02SSLConfig
Management Scope Name (scopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
Client Key Alias (clientKeyAlias): default
Server Key Alias (serverKeyAlias): default
SSL Type (type): [JSSE]
Client Authentication (clientAuthentication): [false]
Security Level of the SSL Configuration (securityLevel): [HIGH]
Enabled Ciphers SSL Configuration (enabledCiphers):
JSSE Provider (jsseProvider): [IBMJSSE2]
Client Authentication Support (clientAuthenticationSupported): [false]
SSL Protocol (sslProtocol): [SSL_TLS]
Trust Manager Object Names (trustManagerObjectNames): (cells/BIRKT40Cell02|security.xml#TrustManager_1)
*Trust Store Name (trustStoreName): NodeDefaultTrustStore
Trust Store Scope (trustStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
*Key Store Name (keyStoreName): NodeDefaultKeyStore
Key Store Scope Name (keyStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
Key Manager Name (keyManagerName): IbmX509
Key Manager Scope Name (keyManagerScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
```

```
Create SSL Configuration

F (Finish)
C (Cancel)

Select [F, C]: [F] F
WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias BIRKT40Node02SSLConfig -scopeName
(cell):BIRKT40Cell02:(node):BIRKT40Node02 -clientKeyAlias default -serverKeyAlias default
-trustManagerObjectNames (cells/BIRKT40Cell02|security.xml#TrustManager_1) -trustStoreName
NodeDefaultTrustStore -trustStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyStoreName
NodeDefaultKeyStore -keyStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyManagerName
IbmX509 -keyManagerScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 }
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Results

The name of the SSL configuration object that you created, for example, (`cells/BIRKT40Cell02|security.xml#SSLConfig_1136652770753`), appears in the `security.xml` file.

Example `security.xml` file output:

```
<repertoire xmi:id="SSLConfig_1136652770753" alias="BIRKT40Node02SSLConfig" type="JSSE"
managementScope="ManagementScope_1134610924357">
<setting xmi:id="SecureSocketLayer_1136652770924" clientKeyAlias="default" serverKeyAlias="default"
clientAuthentication="false" securityLevel="HIGH"  jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1134610924357" trustStore="KeyStore_1134610924377" trustManager="TrustManager_1"
keyManager="KeyManager_1134610924357"/>
</repertoire>
```

What to do next

Once you create the SSL configuration object, the next step is to use it. There are several different ways that you can associate SSL configurations with protocols, for example:

- Set the SSL configuration on the thread programmatically.
- Associate the SSL configuration with an outbound protocol or a target host and port.
- Directly associating the SSL configuration using the alias.
- Centrally managing the SSL configurations by associating them with SSL configuration groups or zones so that they are used based upon the group from where the end point exists.

Automating SSL configurations using scripting

SSL configuration is needed for WebSphere to perform SSL connections with other servers. A SSL configuration can be configured through the Admin Console. But if an automated way to create a SSL configuration is desired then AdminTask should be used.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

AdminTask can be used in a interactive mode and batch mode. For automation the batch mode options should be used. AdminTask batch mode can be called in a JACL or Python script. Interactive mode will step through all the parameter the task needs, requires ones are marked with a '*'. Before the interactive task executes the task it echoes the batch mode syntax of the task to the screen. This can be helpful when writing batch mode scripts.

There attributes needed to create an ssl configurations:

- A key store

- Default client certificate alias
- Default server certificate alias
- Trust store
- The handshake protocol
- The ciphers needed during handshake
- Supporting client authentication or not

If automating the creation of a SSL Configuration it may be needed to create some of the attribute values needed like the key store, trust store, key manager, and trust managers.

Procedure

- To create a SSL configuration the createSSLConfig AdminTask can be used. To make changes to the SSL configurations use the modifySSLConfig AdminTask.
  - Interactive mode:

    Interactive mode steps you through all attributes and tell you the default value of the attribute if there is one. The default value is in '[]' on the prompt line. The actual flag used in batch mode is in '()' on each prompt line. If you are using the default value then the flag will not show up on the batch command line.

    Using Jacl:

```
$AdminTask createSSLConfig -interactive
```

  - Using Jython:

```
AdminTask.createSSLConfig ('[interactive]')
```

    Example output:

```
*SSL Configuration Alias (alias): testSSLConfig
Management Scope Name (scopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Client Key Alias (clientKeyAlias): clientCert
Server Key Alias (serverKeyAlias): serverCert
SSL Type (type): [JSSE]
Client Authentication (clientAuthentication): [false]
Security Level of the SSL Configuration (securityLevel): [HIGH] HIGH
Enabled Ciphers SSL Configuration (enabledCiphers):
JSSE Provider (jsseProvider): [IBMJSSE2]
Client Authentication Support (clientAuthenticationSupported): [false]
SSL Protocol (sslProtocol): [SSL_TLS] SSL_TLS
Trust Manager Object Names (trustManagerObjectNames):
*Trust Store Name (trustStoreName): testTrustStore
Trust Store Scope (trustStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
*Key Store Name (keyStoreName): testKeyStore
Key Store Scope Name (keyStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Key Manager Name (keyManagerName): IbmX509
Key Manager Scope Name (keyManagerScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01

Create SSL Configuration

F (Finish)
C (Cancel)

Select [F, C]: [F]
WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01 }
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

    At the end of the output, the batch mode parameters are provided.

  - Batch mode:

    Using Jacl:

```
$AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01}
```

  - Using Jython:

```
AdminTask.createSSLConfig ('[-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01]')
```

Example output:

```
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

- Key Stores and Trust Stores The key store and trust store may already exist or a new one may need to be created. To create a new key store or trust store use the createKeyStore AdminTask. It will create a key store file and store the configuration object in the system configuration. A trust store is just a key store that usually only has signer certificates in it. To create a key store enter:
  - Using Jacl:

```
$AdminTask createKeyStore {-keyStoreName testKeyStore  -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd  -keyStoreIsFileBased true -keyStoreReadOnly false}
```

  - Using Jython:

```
AdminTask.createKeyStore ('[-keyStoreName testKeyStore  -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd  -keyStoreIsFileBased true -keyStoreReadOnly false]')
```

  To populate the key store with certificates see “Managing Certificates using AdminConsole and Admin Task” The key store and trust store are required to create a SSL configuration. Use the '-keyStoreName' and '-trustStoreName' flags on the createSSLConfig. There scopes can be added with the '-keyStoreScope' flag and '-trustStoreScope' flags.

- Key Manager Key manager are used to determine how a certificate is selected. The IbmX509 key manager is in the security configuration by default. If a different key manager is needed then use createKeyManager AdminTask to create it. To create a key manager enter:
  - Using Jacl:

```
$AdminTask createKeyManager {-name testKeyManager
-scopeName  (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm }
```

  - Using Jython:

```
AdminTask.createKeyManager ('[-name testKeyManager
-scopeName  (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm]')
```

  To supply a key manager on the createSSLConfig AdminTask use the '-keyManagerName' along with the '-keyManagerScope' flag.

- Trust Manager Trust managers are use to determine how trust is established during ssl communication. The IbmX509 and IbmPKIX are trust managers are in the security configuration by default. If a different or additional trust manager is needed then use the createTrustManger AdminTask to create it. To create a trust manager enter:
  - Using Jacl:

```
$AdminTask createTrustManager {-name testTrustManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm }
```

  - Using Jython:

```
AdminTask.createTrustManager ('[-name testTrustManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm]')
```

  The SSL Configuration can have multiple trust managers. To supply multiple trust managers give a comma separated list of the trust managers configuration IDs with the `-trustManagerObjectNames` flag. When you create a trust manager the configuration object ID is returned. To get a list of trust managers object IDs use the **listTrustManagers** command of the AdminTask object with the `-displayObjectName true` flag. For example:

```
wsadmin>$AdminTask listTrustManagers -interactive
List Trust Managers

List trust managers.

Management Scope Name (scopeName):
Display list in ObjectName Format (displayObjectName): [false] true
```

```
List Trust Managers

F (Finish)
C (Cancel)

Select [F, C]: [F]
Inside generate script command
WASX7278I: Generated command line: $AdminTask listTrustManagers {-displayObjectName true }
IbmX509(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_1)
IbmPKIX(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_2)
```

Updating default key store passwords using scripting

Use the Jython or Jacl scripting language to change the default key store passwords. A key store file is created with a default password when you install the application server. Change this password to protect your security configuration.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

When you install the application server, each server creates a key store and trust store for the default SSL configuration with the default password `WebAS`. To protect the security of the key store files and the SSL configuration, you must change the password. The following examples update the default password:

Procedure

- Change multiple key stores passwords. The **changeMultipleKeyStorePasswords** command updates all of the key stores that have the same password. For example:
  - Using Jacl:

```
$AdminTask changeMultipleKeyStorePasswords  {-keyStorePassword WebAS
-newKeyStorePassword  secretPwd   -newKeyStorePasswordVerify secretPwd}
```

  - Using Jython:

```
AdminTask.changeMultipleKeyStorePasswords  ('[-keyStorePassword WebAS
-newKeyStorePassword secretPwd -newKeyStorePasswordVerify secretPwd]')
```

- Change the password of a single key store. The **changeKeyStorePassword** command updates the password of an individual key store. For example:
  - Using Jacl:

```
$AdminTask changeKeyStorePassword {-keyStoreName testKS
-scopeName (cell):localhost:(server):server1
-keyStorePassword WebAS -newKeyStorePassword secretPwd
-newKeyStorePasswordVerify secretPwd}
```

  - Using Jython:

```
AdminTask.changeKeyStorePassword ('[-keyStoreName testKS
-scopeName (cell):localhost:(server):server1
-keyStorePassword WebAS -newKeyStorePassword secretPwd
-newKeyStorePasswordVerify secretPwd]')
```

Configuring certificate authority client objects using the wsadmin tool

Use this topic to create a certificate authority (CA) client object. The client object contains all of the configuration information necessary to connect to your third-party CA server. A CA client must exist in your configuration before you can issue a request to the CA to create personal certificates with the requestCACertificate command.

Before you begin

A CA client object contains information that the system uses to connect to a certificate authority. Implement the com.ibm.ws.WSPKIClient interface to connect to the certificate authority and provide the com.ibm.ws.WSPKIClient class when creating the CA client object.

About this task

If a CA client does not exist in your configuration, use the steps in this topic to create a new CA client.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Determine if a CA client exists in your configuration.

   Use the following listCAClients command to list all certificate authority clients in your configuration:

```
print AdminTask.listCAClients()
```

3. If no CA clients exist, then create a new CA client.

   Use the createCAClient command to create a new CA client object. The application server connects to a CA server through the WSPKIClient() implementation, which handles all connections and communications with the CA server.

Table 30. Required parameter. You must specify the following configuration information for a new CA client object:

Parameter	Description	Data Type
-CAClientName	Specify a name to uniquely identify the CA client object.	String

Table 31. Additional parameters. You can specify additional configuration information using the following parameters:

Parameter	Description	Data Type
-scopeName	Specify the management scope of the CA client. For a deployment manager profile, the system uses the cell scope as the default value. For an application server profile, the system uses the node scope as the default value.	String
-pkiClientImplClass	Specify the class path that implements the WSPKIClient interface. The system uses this path to connect to the CA and to issue requests to the CA. The default value is `com.ibm.wsspi.ssl.WSPKIClient`.	String
-host	Specify the host name in your system where the CA resides.	String
-port	Specify the port on the server where the CA listens.	String
-userName	Specify the user name to use to authenticate to the CA.	String
-password	Specify the password for the user name that authenticates to the CA.	String
-frequencyCheck	Specify how often, in minutes, the system checks with the CA to determine if a certificate has been created.	String
-retryCheck	Specify the number of times to check with the CA to determine if a certificate has been created.	String
-customProperties	Specifies a comma separated list of attribute and value (attribute=value) custom property pairs to add to the CA client object.	String

   Use the following example command to create a new CA client object:

```
AdminTask.createCAClient('[-caClientName clientObj01 -pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient
-host machine011 -port 9022 -userName admin -password pw4admin]')
```

   The command returns the object name of the CA client that has been created.

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

What to do next

If the CA client object was successfully created, then you can configure the application server to use a personal certificate created by an external CA.

Administering certificate authority clients using the wsadmin tool

Use this topic to modify certificate authority (CA) client objects. The client object contains all of the configuration information necessary to connect to your third-party CA server.

Before you begin

You must configure a CA client object in your environment.

About this task

For existing CA client objects, use the steps in this topic to view, modify, or delete existing CA client object configurations.

Procedure

- View existing CA client objects and configuration data. Use the listCAClients and getCAClient commands to query your environment for your existing CA clients.
  1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
  2. List all CA client objects in your configuration.

     Use the listCAClients command to list all certificate authority clients in your configuration. If you do not provide a value for the -scopeName parameter, then the command queries the cell if you use a deployment manager profile or queries the node if you use an application server profile. Use the -all parameter to query your environment without using a specific scope, as the following example demonstrates:

```
print AdminTask.listCAClients('-all true')
```

     The command returns an array of attribute lists, displaying one attribute list for each CA client, as the following example output displays:

```
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
] [scopeName (cell):myCell01] [name jenCAClient] [baseDn ] [_Websphere_Config_Da
ta_Id cells/myCell01|security.xml#CAClient_1181834566881] [port 2950] [CACertifi
cate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Webspher
e_Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pa
ssword ] [host ] ]'
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphere_Config_Dat
a_Id cells/myCell01|security.xml#CAClient_1181834566882] [port 2951] [CACertific
ate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Websphere
_Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pas
sword ] [host ] ]'
```

  3. List the configuration attributes for a specific CA client.

     Use the getCAClient command to view the list of attributes for a specific CA client, as the following example demonstrates:

```
print AdminTask.getCAClient('-caClientName myCAClient')
```

     The command returns an attribute list that contains the attribute and value pairs for the specific CA client, as the following example demonstrates:

```
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementSc
ope_1)] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphe
re_Config_Data_Id cells/myCell01|security.xml#CAClient_1181834566882] [por
t 2951] [CACertificate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [u
serId ] [_Websphere_Config_Data_Type CAClient] [retryCheck 0] [properties ] [fre
quencyCheck 0] [password ] [host ] ]'
```

- Modify your existing CA client object configuration data. Use the modifyCAClient command to change one or more configuration attributes for a specific CA client.
  1. Start the wsadmin scripting tool.

2. Determine which configuration attributes to edit.

   The modifyCAClient modifies all attributes that you specify with the command parameters. If you do not specify a parameter, then its corresponding attribute does not change.

Table 32. Command parameters. You can edit the following configuration data with the modifyCAClient command:

Parameter	Description	Data Type
-scopeName	Specify the management scope of the CA client. For a deployment manager profile, the system uses the cell scope as the default. For an application server profile, the system uses the node scope as the default.	String
-pkiClientImplClass	Specify the class path that implements the WSPKIClient interface. The system uses this path to connect to the CA and to issue requests to the CA.	String
-host	Specify the host name in your system where the CA resides.	String
-port	Specify the port on the server where the CA listens.	String
-userName	Specify the user name to use to authenticate to the CA.	String
-password	Specify the password for the user name that authenticates to the CA.	String
-frequencyCheck	Specify how often, in minutes, the system should check with the CA to determine if a certificate has been created.	String
-retryCheck	Specify the number of times to check with the CA to determine if a certificate has been created.	String
-customProperties	Specifies a comma separated list of attribute and value (attribute=value) custom property pairs to modify on the CA Client object. You can create, modify, or remove properties. To remove a property specify `attribute=` attribute as equal to no value.	String

3. Modify specific configuration attributes for a CA client object.

   Use the following example command to modify the port number of the CA, the user name, and password attributes for the `myCAClient` CA client object:

```
AdminTask.modifyCAClient('[-caClientName myCAClient -port 4060 -userName admin
 -password password4admin -pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient]')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

- Remove a CA client object from your configuration. Use the deleteCAClient command to delete a CA client object from your configuration. The command does not delete the CA client object if the CA client to delete is referenced by a certificate object.
  1. Start the wsadmin scripting tool.
  2. Determine the CA client object to delete.

     Use the listCAClients command to list all certificate authority clients in your configuration. If you do not provide a value for the -scopeName parameter, then the command queries the cell if you use a deployment manager profile or queries the node if you use an application server profile. Use the -all parameter to query your environment without using a specific scope, as the following example demonstrates:

```
print AdminTask.listCAClients('-all true')
```

  3. Delete the CA client object of interest.

     Use the deleteCAClient command to delete the CA client object from your configuration. Use the -caClientName parameter to specify the CA client to delete. You can optionally specify the

management scope of the CA client object with the scopeName parameter. The following example command removes the `myCAClient` CA client object:

```
AdminTask.deleteCAClient('[-caClientName myCAClient]')
```

If you receive an error message, then verify that the CA client object of interest exists in your configuration and that it is not referenced by a certificate object in your security configuration.

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

Setting a certificate authority certificate as the default certificate using the wsadmin tool

Use this topic to make a request to an external certificate authority (CA) to create a personal certificate. After the CA returns the certificate and the certificate is saved in the keystore, then you can use it as the server default personal certificate.

Before you begin

You must configure a CA client object in your environment. The client object contains all of the configuration information necessary to connect to your third-party CA server.

About this task

After profile creation, the system is assigned a default chained personal certificate. Use the following steps to modify the application server to use a default personal certificate created by an external CA.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Verify that a certificate authority client exists in your configuration. Use the listCAClients command to query your environment for all existing certificate authority clients and configuration attributes, or the getCAClient command to return the configuration attributes for a specific certificate authority client. If the listCAClients or getCAClient commands do not return any attributes, then you must create a certificate authority client object before you can complete the remaining steps.
   - List all certificate authority client objects in your configuration.

     Use the listCAClients command to list all certificate authority clients in your configuration. If you do not provide a value for the -scopeName parameter, then the command queries the cell if you use a deployment manager profile or queries the node if you use an application server profile. Use the -all parameter to query your environment without using a specific scope, as the following example demonstrates:

     ```
     print AdminTask.listCAClients('-all true')
     ```

     The command returns an array of attribute lists, displaying one attribute list for each CA client, as the following example output displays:

     ```
     '[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
     ] [scopeName (cell):myCell01] [name jenCAClient] [baseDn ] [_Websphere_Config_Da
     ta_Id cells/myCell01|security.xml#CAClient_1181834566881] [port 2950] [CACertifi
     cate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Webspher
     e_Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pa
     ssword ] [host ] ]'
     '[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
     ] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphere_Config_Dat
     a_Id cells/myCell01|security.xml#CAClient_1181834566882] [port 2951] [CACertific
     ate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Websphere
     _Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pas
     sword ] [host ] ]'
     ```

   - List the configuration attributes for a specific certificate authority client.

     Use the getCAClient command to view the list of attributes for a specific certificate authority client, as the following example demonstrates:

```
print AdminTask.getCAClient('-caClientName myCAClient')
```

The command returns an attribute list that contains the attribute and value pairs for the specific certificate authority client, as the following example demonstrates:

```
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementSc
ope_1)] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphe
re_Config_Data_Id cells/myCell01|security.xml#CAClient_1181834566882] [por
t 2951] [CACertificate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [u
serId ] [_Websphere_Config_Data_Type CAClient] [retryCheck 0] [properties ] [fre
quencyCheck 0] [password ] [host ] ]'
```

3. Optional: If a certificate authority client does not exist in your environment, then configure a CA client object.
4. Optional: View the current default personal certificate.

   Use the following listPersonalCertificates command to display the current default personal certificate to replace:

```
AdminTask.listPersonalCertificates('[-keyStoreName CellDefaultKeyStore -keyStoreScope (cell):myCell01]')
```

5. Request a certificate from a certificate authority.

   Before the current default personal certificate can be replaced, you must request a certificate from a certificate authority. You can create a new certificate request or use the createCertificateRequest command to use a predefined certificate request. The system uses the certificate request and the certificate authority configuration information from the CA client object to request the certificate from the certificate authority. If the certificate authority returns a certificate, then the requestCAcertificate command stores the certificate in the specified key store and returns a message of `COMPLETE`.

Table 33. Required parameters. Use the requestCACertificate command and the following required parameters to request a certificate from a certificate authority:

Parameter	Description	Data Type
-certificateAlias	Specifies the alias of the certificate. You can specify a predefined certificate request.	String
-keyStoreName	Specifies the name of the keystore object that stores the CA certificate. Use the listKeyStores command to display a list of available keystores.	String
-caClientName	Specifies the name of the CA client that was used to create the CA certificate.	String
-revocationPassword	Specifies the password to use to revoke the certificate at a later date.	String

Table 34. Optional parameters. You can also use the following parameters to specify additional certificate request options. If you do not specify an optional parameter, then the command uses the default value.

Parameter	Description	Data Type
-keyStoreScope	Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope.	String
-caClientScope	Specifies the management scope of the CA client. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope.	String
-certificateCommonName	Specifies the common name (CN) part of the full distinguished name (DN) of the certificate. This common name can represent a person, company, or machine. For websites, the common name is frequently the DNS host name where the server resides.	String
-certificateSize	Specifies the size of the certificate key. The valid values are `512`, `1024`, `2048`, `4096` and `8192`. The default value is `2048`.	String
-certificateOrganization	Specifies the organization portion of the distinguished name.	String
-certificateOrganizationalUnit	Specifies the organizational unit portion of the distinguished name.	String

Table 34. Optional parameters (continued). You can also use the following parameters to specify additional certificate request options. If you do not specify an optional parameter, then the command uses the default value.

Parameter	Description	Data Type
-certificateLocality	Specifies the locality portion of the distinguished name.	String
-certificateState	Specifies the state portion of the distinguished name.	String
-certificateZip	Specifies the zip code portion of the distinguished name.	String
-certificateCountry	Specifies the country portion of the distinguished name.	String

Use the following example command syntax to request a certificate from a certificate authority:

```
AdminTask.requestCACertificate('-certificateAlias newCertificate -keyStoreName
CellDefaultKeyStore -caClientName myCAClient -revocationPassword revokeCApw
-pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient')
```

The command returns one of two values: `Certificate COMPLETE` or `certificate PENDING`. If the command returns the `Certificate COMPLETE` message, the certificate authority returned the requested certificate and the default personal certificate is replaced. If the command returns the `certificate PENDING` message, the certificate authority did not yet return a certificate. Use the queryCACertificate command to view the current status of the certificate request, as the following example demonstrates:

```
AdminTask.queryCACertificate('-certificateAlias newCertificate -keyStoreName
CellDefaultKeyStore -pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient')
```

6. Replace the server default personal certificate.

   Use the following replaceCertificate command example to replace the existing default personal certificate with the newly created CA personal certificate:

```
AdminTask.replaceCertificate('-keyStoreName CellDefaultKeyStore -certificateAlias
defaultPersonalCertificate -replacementCertificateAlias newCertificate')
```

7. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Results

The default personal certificate for the server is a certificate that is created by an external CA.

What to do next

If the CA client object was successfully created, then you can configure the application server to use a personal certificate created by an external CA.

Creating certificate authority (CA) personal certificates using the wsadmin tool

Use this topic to create CA certificates from a certificate authority (CA).

Before you begin

You must configure a CA client object in your environment. The client object contains all of the configuration information necessary to connect to your third-party CA server.

About this task

Use the following information to create a CA personal certificate using a CA client.

Procedure

1. Optional: Query your configuration for keystores to determine where system stores the new CA certificate.

Use the listKeyStores command to list all keystores for a specific management scope. Specify the -scopeName parameter to display keystores within a specific management scope, or set the -all parameter to `true` to display all keystores regardless of scope. The following example lists all keystores in your configuration:

```
AdminTask.listKeystores('-all true')
```

The command returns the following sample output:

```
CellDefaultKeyStore(cells/myCell|security.xml#KeyStore_1)
CellDefaultTrustStore(cells/myCell|security.xml#KeyStore_2)
CellLTPAKeys(cells/myCell|security.xml#KeyStore_3)
NodeDefaultKeyStore(cells/myCell|security.xml#KeyStore_1598745926544)
NodeDefaultTrustStore(cells/myCell|security.xml#KeyStore_1476529854789)
```

Use the getKeyStoreInfo command and specify the -keyStoreName parameter to return additional information about the keystore of interest, as the following example displays:

```
AdminTask.getKeyStoreInfo('[-keyStoreName CellDefaultKeyStore]')
```

The command returns the following configuration information for the keystore of interest:

```
[ [location ${CONFIG_ROOT}/cells/myCell/key.p12] [password *****] [_Webspher
e_Config_Data_Id cells/myCell|security.xml#KeyStore_1] [_Websphere_Config_Da
ta_Version ] [useForAcceleration false] [slot 0] [type PKCS12] [additionalKeySto
reAttrs ] [fileBased true] [_Websphere_Config_Data_Type KeyStore] [customProvide
rClass ] [hostList ] [createStashFileForCMS false] [description [Default key sto
re for JenbCell01]] [readOnly false] [initializeAtStartup false] [managementScop
e (cells/JenbCell01|security.xml#ManagementScope_1)] [usage SSLKeys] [provider I
BMJCE] [name CellDefaultKeyStore] ]
```

2. Optional: Determine which CA client to use.

   Use the listCAClients command to list the CA clients that exist in your configuration. Specify the -scopeName parameter to display CA clients within a specific management scope, or set the -all parameter to `true` to display all CA clients regardless of scope. The following example lists all CA clients in your configuration:

```
AdminTask.listCAClients('-all true')
```

3. Create a CA personal certificate.

   Use the requestCACertificate command to create a new CA personal certificate in your environment. The system uses the certificate request and the certificate authority configuration information from the CA client object to request the certificate from the certificate authority. If the certificate authority returns a certificate, the requestCAcertificate command stores the certificate in the specified key store and returns a message of `COMPLETE`.

Table 35. Required parameters. Use the requestCACertificate command and the following required parameters to request a certificate from a certificate authority:

Parameter	Description	Data type
-certificateAlias	Specifies the alias of the certificate. You can specify a predefined certificate request.	String
-keyStoreName	Specifies the name of the keystore object that stores the CA certificate. Use the listKeyStores command to display a list of available keystores.	String
-caClientName	Specifies the name of the CA client that was used to create the CA certificate.	String
-revocationPassword	Specifies the password to use to revoke the certificate at a later date.	String

Table 36. Additional parameters. You can also use the following parameters to specify additional certificate request options. If you do not specify an optional parameter, the command uses the default value.

Parameter	Description	Data type
-keyStoreScope	Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope.	String

Table 36. Additional parameters (continued). You can also use the following parameters to specify additional certificate request options. If you do not specify an optional parameter, the command uses the default value.

Parameter	Description	Data type
-caClientScope	Specifies the management scope of the CA client. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope.	String
-certificateCommonName	Specifies the common name (CN) part of the full distinguished name (DN) of the certificate. This common name can represent a person, company, or machine. For websites, the common name is frequently the DNS host name where the server resides.	String
-certificateSize	Specifies the size of the certificate key. The valid values are 512, 1024, 2048, 4096 and 8192. The default value is 2048.	String
-certificateOrganization	Specifies the organization portion of the distinguished name.	String
-certificateOrganizationalUnit	Specifies the organizational unit portion of the distinguished name.	String
-certificateLocality	Specifies the locality portion of the distinguished name.	String
-certificateState	Specifies the state portion of the distinguished name.	String
-certificateZip	Specifies the zip code portion of the distinguished name.	String
-certificateCountry	Specifies the country portion of the distinguished name.	String

Use the following example command syntax to request a certificate from a certificate authority:

```
AdminTask.requestCACertificate('-certificateAlias newCertificate -keyStoreName CellDefaultKeyStore
-CAClientName myCAClient -revocationPassword revokeCApw')
```

The command returns one of two values: `Certificate COMPLETE` or `certificate PENDING`. If the command returns the `Certificate COMPLETE` message, the certificate authority returned the requested certificate and the default personal certificate is replaced. If the command returns the `certificate PENDING` message, the certificate authority did not yet return a certificate. Use the queryCACertificate command to view the current status of the certificate request, as the following example displays:

```
AdminTask.queryCACertificate('-certificateAlias newCertificate -keyStoreName CellDefaultKeyStore')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

Results

The default personal certificate for the server is a certificate that is created by an external CA.

Revoking certificate authority personal certificates using the wsadmin tool

You can revoke CA certificates from a certificate authority (CA). Revoke personal certificates that are no longer being used in your configuration.

Before you begin

Use the requestCACertificate command to create a personal certificate with the requestCACertificate task before you can request that the certificate authority revoke the certificate. Certificates created with the requestCACertificate command have an associated reference object in the configuration that you can use to submit the certificate revocation request to the certificate authority.

About this task

This topic uses the revokeCACertificate command to submit a request to revoke a certificate on the certificate authority. You can only revoke a certificate that was created with the requestCACertificate command. You must specify the revocation password that was provided when the certificate was created. Use the same password to revoke the certificate on the certificate authority.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Determine the CA personal certificate to revoke.

   Use the listPersonalCertificates command to view a list of all personal certificates and associated attributes for a specific keystore, as the following example demonstrates:

```
AdminTask.listPersonalCertificates('-keyStoreName CellDefaultKeyStore')
```

   The command returns an attribute list for each personal certificate, including CA personal certificates. CA personal certificates only return the status attribute. You can revoke each CA personal certificates that returns a `COMPLETE` status. Determine which CA personal certificate to revoke.
3. Revoke a CA personal certificate.

   Use the revokeCACertificate command to revoke the CA personal certificate of interest. You must specify the name of the keystore, certificate alias, and revocation password using the following parameters:

Table 37. Required parameters. This table describes the revokeCACertificate command and its optional parameters:

Parameter	Description	Data Type
-keyStoreName	Specifies the name of the keystore where the CA personal certificate is stored. The value of this field is not a path to the keystore file.	String
-certificateAlias	Specifies the unique name that identifies the CA personal certificate object and the alias name of the certificate in the keystore.	String
-revocationPassword	Specifies the password needed to revoke the certificate. This is the same password that was provided when the certificate was created.	String

   You can specify additional information with the following optional parameters:

Table 38. Optional parameters. This table describes the revokeCACertificate command and its additional optional parameters

Parameter	Description	Data Type
-keyStoreScope	Specifies the management scope of the keystore. For a deployment manager profile, the system uses the cell scope as the default value. For an application server profile, the system uses the node scope as the default value. To obtain a list of the keystore scope values, see the listManagementScopes command, which is part of the ManagementScopeCommands command group.	String
-revocationReason	Specifies the reason for revoking the certificate of interest. The default value for this parameter is `unspecified`.	String

   The following example revokes a CA personal certificate:

```
AdminTask.revokeCACertificate('[-keyStoreName CellDefaultKeyStore -certificateAlias myCertificate -revocationPassword pw4revoke]')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

CAClientCommands command group for the AdminTask object

You can use the Jython scripting language to manage your certificate authority (CA) client configurations with the wsadmin tool. Use the commands and parameters in the CAClientCommands group to create, modify, query, and remove connections to a third-party CA server.

Use the following commands to manage your certificate authority (CA) client configurations:

- "createCAClient"
- "modifyCAClient" on page 193
- "getCAClient" on page 194
- "deleteCAClient" on page 195
- "listCAClients" on page 195

createCAClient

The createCAClient command creates a new CA client object in your configuration. The application server connects to a CA server through the WSPKIClient() implementation, which handles all connections and communications with the CA server.

Target object

None.

Required parameters

-caClientName
Specifies a name to uniquely identify the CA client object. (String, required)

-pkiClientImplClass
Specifies the class path that implements the WSPKIClient interface. The system uses this path to connect to the CA and to issue requests to the CA. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the CA client. For a deployment manager profile, the system uses the cell scope as the default value. For an application server profile, the system uses the node scope as the default value. (String, optional)

-host
Specifies the host name in your system where the CA resides. (String, optional)

-port
Specifies the port on the server where the CA listens. (String, optional)

-userName
Specifies the user name to use to authenticate to the CA. (String, optional)

-password
Specifies the password for the user name that authenticates to the CA. (String, optional)

-frequencyCheck
Specifies how often, in minutes, the system communicates with the CA to determine if a certificate has been created. (String, optional)

-retryCheck
Specifies the number of times to communicate with the CA to determine if a certificate has been created. (String, optional)

-customProperties
Specifies a comma-separated list of attribute and value custom property pairs to add to the CA client object, using the following format: `attribute=value,attribute=value`. (String, optional)

Return value

The command returns the object name of the CA client that the system creates.

Batch mode example usage

- Using Jython string:

```
AdminTask.createCAClient('[-caClientName clientObj01 -pkiClientImplClass
com.ibm.wsspi.ssl.WSPKIClient -host machine011 -port 9022
-userName admin -password pw4admin]')
```

- Using Jython list:

```
AdminTask.createCAClient(['-caClientName', 'clientObj01', '-pkiClientImplClass',
'com.ibm.wsspi.ssl.WSPKIClient', '-host', 'machine011', '-port', '9022',
'-userName', 'admin', '-password', 'pw4admin'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createCAClient('-interactive')
```

modifyCAClient

The modifyCAClient command modifies your existing CA client object configuration data. You can modify one or multiple configuration attributes for a specific CA client.

Target object

None.

Required parameters

-caClientName
Specifies the name of the CA client of interest. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the CA client. For a deployment manager profile, the system uses the cell scope as the default. For an application server profile, the system uses the node scope as the default. (String, optional)

-pkiClientImplClass
Specifies the class path that implements the WSPKIClient interface. The system uses this path to connect to the CA and to issue requests to the CA. (String, optional)

-host
Specifies the host name in your system where the CA resides. (String, optional)

-port
Specifies the port on the server where the CA listens. (String, optional)

-userName
Specifies the user name to use to authenticate to the CA. (String, optional)

-password
Specifies the password for the user name that authenticates to the CA. (String, optional)

-frequencyCheck
Specifies how often, in minutes, the system should check with the CA to determine if a certificate has been created. (String, optional)

-retryCheck
Specifies the number of times to check with the CA to determine if a certificate has been created. (String, optional)

-customProperties
Specifies a comma separated list of attribute and value (attribute=value) custom property pairs to modify on the CA Client object. You can create, modify, or remove properties. To remove a property specify the attribute and value as `attribute=`. (String, optional)

Return value

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyCAClient('[-caClientName myCAClient -port 4060
-userName admin -password password4admin]')
```

- Using Jython list:

```
AdminTask.modifyCAClient(['-caClientName', 'myCAClient', '-port', '4060',
'-userName', 'admin', '-password', 'password4admin'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyCAClient('-interactive')
```

getCAClient

The getCAClient command displays a list of attributes for a specific CA client.

Target object

None.

Required parameters

-caClientName
Specifies the CA client name of interest. (String, required)

Optional parameters

-scopeName
Specifies the management scope of CA client of interest. (String, optional)

Return value

The command returns an attribute list that contains the attribute and value pairs for the specific CA client, as the following example displays:

```
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementSc
ope_1)] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphe
re_Config_Data_Id cells/myCell01|security.xml#CAClient_1181834566882] [por
t 2951] [CACertificate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [u
serId ] [_Websphere_Config_Data_Type CAClient] [retryCheck 0] [properties ] [fre
quencyCheck 0] [password ] [host ] ]'
```

Batch mode example usage

- Using Jython string:

```
print AdminTask.getCAClient('-caClientName myCAClient')
```

- Using Jython list:

```
print AdminTask.getCAClient('-caClientName', 'myCAClient')
```

Interactive mode example usage

- Using Jython string:

```
print AdminTask.getCAClient('-interactive')
```

deleteCAClient

The deleteCAClient command removes the CA client object of interest from your configuration. Use the -caClientName parameter to specify the CA client to delete. You can optionally specify the management scope of the CA client object with the scopeName parameter.

Target object

None.

Required parameters

-caClientName
Specifies the name of the CA client of interest. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the CA client of interest. (String, optional)

Return value

The command does not return output if the system successfully removes the CA client of interest. If you receive an error message, verify that the CA client object of interest exists in your configuration and that it is not referenced by a certificate object in your security configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteCAClient('[-caClientName myCAClient]')
```

- Using Jython list:

```
AdminTask.deleteCAClient(['-caClientName', 'myCAClient'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteCAClient('-interactive')
```

listCAClients

The listCAClients command lists all CA clients in your configuration or within a specific scope. If you do not provide a value for the -scopeName parameter, the command queries the cell if you use a deployment manager profile or queries the node if you use an application server profile. Use the -all parameter to query your environment without using a specific scope.

Target object

None.

Optional parameters

-scopeName
Specifies the management scope to search for CA clients. (String, optional)

-all
Specifies whether the system queries for CA clients without a specific scope. (Boolean, optional)

Return value

The command returns an array of attribute lists, displaying one attribute list for each CA client, as the following example output displays:

```
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
] [scopeName (cell):myCell01] [name jenCAClient] [baseDn ] [_Websphere_Config_Da
ta_Id cells/myCell01|security.xml#CAClient_1181834566881] [port 2950] [CACertifi
cate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Webspher
e_Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pa
ssword ] [host ] ]'
'[ [backupCAs ] [managementScope (cells/myCell01|security.xml#ManagementScope_1)
] [scopeName (cell):myCell01] [name myCAClient] [baseDn ] [_Websphere_Config_Dat
a_Id cells/myCell01|security.xml#CAClient_1181834566882] [port 2951] [CACertific
ate ] [pkiClientImplClass com.ibm.wsspi.ssl.WSPKIClient] [userId ] [_Websphere
_Config_Data_Type CAClient] [retryCheck 0] [properties ] [frequencyCheck 0] [pas
sword ] [host ] ]'
```

Batch mode example usage

- Using Jython string:

```
print AdminTask.listCAClients('-all true')
```

- Using Jython list:

```
print AdminTask.listCAClients('-all', 'true')
```

Interactive mode example usage

- Using Jython:

```
print AdminTask.listCAClients('-interactive')
```

Creating self-signed certificates using scripting

Use the Jython or Jacl scripting language to create self-signed certificates with the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

You can create self-signed certificates using the scripting and the AdminTask object. You can run the commands in interactive or batch mode. Interactive mode provides a way to discover the flags that you need to run the task in batch mode.

Certificates reside inside of key stores. To run the commands, you will need the name of the key store to be supplied. Use the **listKeyStore** command of the AdminTask object to get a list of key stores. If you need a new key store, use the **createKeyStore** command of the AdminTask object.

To create a personal key store, use the following examples:

Procedure

- Interactive mode:
  - Using Jython:

```
AdminTask.createSelfSignedCertificate ('[-interactive]')
```

  - Using Jacl:

```
$AdminTask createSelfSignedCertificate -interactive
```

Example output:

```
*Key Store Name (keyStoreName): keyStore
Key Store Scope Name (keyStoreScope):
*Certificate Alias (certificateAlias): newCert
"Certificate Version" (certificateVersion): 3
*Key Size (certificateSize): [1024]
*Common Name (certificateCommonName): localhost
*Organization (certificateOrganization): workgroup
Organizational Unit (certificateOrganizationalUnit): testing
certLocality (certificateLocality): austin
State (certificateState): Texas
Zip (certificateZip): 78757
Country (certificateCountry): [US]
Validity Period (certificateValidDays): [365]
Create Self-Signed Certificate

F (Finish)
C (Cancel)

Select [F, C]: [F]

WASX7278I: Generated command line: $AdminTask createSelfSignedCertificate
{-keyStoreName keyStore -certificateAlias newCert -certificateVersion 3
-certificateCommonName localhost -certificateOrganization ibm
-certificateOrganizationalUnit testing -certificateLocality austin
-certificateState Texas -certificateZip 78757 }
true
```

At the end of the output, the batch mode parameters are provided.

- Batch mode:
  - Using Jython:

```
AdminTask.createSelfSignedCertificate ('[-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateSize 1024
-certificateCommonName localhost -certificateOrganization ibm
-certificateOrganizationalUnit testing -certificateLocality austin
-certificateState Texas -certificateZip 78757]')
```

  - Using Jacl:

```
$AdminTask createSelfSignedCertificate {-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateSize 1024
-certificateCommonName localhost -certificateOrganization ibm
-certificateOrganizationalUnit testing -certificateLocality austin
-certificateState Texas -certificateZip 78757 }
```

keyManagerCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security. The commands and parameters in the keyManagerCommands group can be used to manage key manager settings. You can use these commands to create, modify, list, or obtain information about key managers.

The keyManagerCommands command group for the AdminTask object includes the following commands:

- "createKeyManager"
- "deleteKeyManager" on page 198
- "getKeyManager" on page 199
- "listKeyManagers" on page 199
- "modifyKeyManager" on page 200

createKeyManager

The **createKeyManager** command creates a key manager in the configuration.

Target object

None

Parameters and return values

-name

The name that uniquely identifies the key manager. (String, required)

-scopeName
The name of the scope. (String, optional)

-provider
Specifies the provider. (String, optional)

-algorithm
Specifies the algorithm name of the key manager. (String, optional)

-keyManagerClass
Specifies the custom class that implements the KeyManager interface. (String, optional)

Examples

Batch mode example usage:

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createKeyManager {-interactive}
```

- Using Jython string:

```
AdminTask.createKeyManager ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createKeyManager (['-interactive'])
```

deleteKeyManager

The **deleteKeyManager** command deletes the key manager settings from the configuration.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key manager. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyManager {-name testKM}
```

- Using Jython string:

```
AdminTask.deleteKeyManager('[-name testKM]')
```

- Using Jython list:

```
AdminTask.deleteKeyManager(['-name', 'testKM'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyManager {-interactive}
```

- Using Jython:

```
AdminTask.deleteKeyManager('-interactive')
```

getKeyManager

The **getKeyManager** command displays a properties object that contains the key manager attributes and values.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key manager. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getKeyManager {-name testKM}
```

- Using Jython string:

```
AdminTask.getKeyManager('[-name testKM]')
```

- Using Jython list:

```
AdminTask.getKeyManager(['-name', 'testKM'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getKeyManager {-interactive}
```

- Using Jython:

```
AdminTask.getKeyManager('-interactive')
```

listKeyManagers

The **listKeyManagers** command lists the key managers within a particular management scope.

Target object

None.

Required parameters

None.

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

-displayObjectName
Set the value of this parameter to `true` to list the key manager objects within the scope. Set the value of this parameter to `false` to list the strings that contain the key manager name and the management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all key managers. This parameter overrides the scopeName parameter. The default value is `false`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeyManagers
```

- Using Jython:

```
AdminTask.listKeyManagers()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeyManagers {-interactive}
```

- Using Jython:

```
AdminTask.listKeyManagers('-interactive')
```

modifyKeyManager

The **modifyKeyManager** command changes existing key manager settings.

Target object

None.

Required parameters

-name
The name that uniquely identifies the key manager. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

-provider
Specifies the provider name of the key manager. (String, optional)

-algorithm
Specifies the algorithm name of the key manager. (String, optional)

-keyManagerClass
Specifies the name of the key manager implementation class. You cannot use this parameter with the provider or the algorithm parameter. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifyKeyManager {-name testKM -provider IBMJSSE2 -algorithm IbmX509}
```

- Using Jython string:

```
AdminTask.modifyKeyManager('[-name testKM -provider IBMJSSE2 -algorithm IbmX509]')
```

- Using Jython list:

```
AdminTask.modifyKeyManager(['-name', 'testKM', '-provider', 'IBMJSSE2', '-algorithm', 'IbmX509'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyKeyManager {-interactive}
```

- Using Jython:

```
AdminTask.modifyKeyManager('-interactive')
```

KeyStoreCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure keystores with the wsadmin tool. A keystore is created by the application server during install and can contain cryptographic keys or certificates. The commands and parameters in the KeyStoreCommands group can be used to create, delete, and manage keystores.

The KeyStoreCommands command group for the AdminTask object includes the following commands:

changeKeyStorePassword

The **changeKeyStorePassword** command modifies the password of a keystore. The command automatically saves the new password to the configuration.

Required parameters

-keyStoreName
Specifies the name of the password to change. (String, required)

-keyStorePassword
Specifies the name of the password to change. (String, required)

-newKeyStorePassword
Specifies the new password that to use to access the keystore. (String, required)

-newKeyStorePasswordVerify
Specifies the new password to confirm the new keystore password. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the keystore. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask changeKeyStorePassword {-keystoreName myKeystore -keyStorePassword
WebAS -newKeyStorePassword newpwd -newKeyStorePasswordVerify newpwd}
```

- Using Jython string:

```
AdminTask.changeKeyStorePassword('[-keystoreName myKeystore -keyStorePassword
WebAS -newKeyStorePassword newpwd -newKeyStorePasswordVerify newpwd]')
```

- Using Jython list:

```
AdminTask.changeKeyStorePassword(['-keystoreName', 'myKeystore', '-keyStorePassword',
'WebAS', '-newKeyStorePassword', 'newpwd', '-newKeyStorePasswordVerify', 'newpwd'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask changeKeyStorePassword {-interactive}
```

- Using Jython:

```
AdminTask.changeKeyStorePassword('-interactive')
```

changeMultipleKeyStorePasswords

The **changeMultipleKeyStorePasswords** command updates the passwords for each keystores in the configuration that has a specific password. This is useful because when you create keystore files on the system, they will have WebAS as a password by default.

Required parameters

-keyStorePassword
Specifies the name of the password that you want to change. (String, required)

-newKeyStorePassword
Specifies the new password that you will use to access the keystore. (String, required)

-newKeyStorePasswordVerify
Confirms the new keystore password. (String, required)

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask changeMultipleKeyStorePasswords {-keyStorePassword WebAS
-newKeyStorePassword newpwd -newKeyStorePasswordVerify newpwd}
```

- Using Jython string:

```
AdminTask.changeMultipleKeyStorePasswords('[-keyStorePassword WebAS
-newKeyStorePassword newpwd -newKeyStorePasswordVerify newpwd]')
```

- Using Jython list:

```
AdminTask.changeMultipleKeyStorePasswords(['-keyStorePassword', 'WebAS',
'-newKeyStorePassword', 'newpwd', '-newKeyStorePasswordVerify', 'newpwd'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask changeMultipleKeyStorePasswords {-interactive}
```

- Using Jython:

```
AdminTask.changeMultipleKeyStorePasswords('-interactive')
```

createKeyStore

The **createKeyStore** command creates the keystore settings in the configuration and the keystore database.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-keyStoreType
The implementation of the keystore management. (String, required)

-keyStoreLocation
The location of the keystore. For file based, the location is the files system path to the keystore database. For hardware keystore, the location is the path to the token library. (String, required)

If you create the IBMi5OSKeyStore keystore, the keystore location must include the `.kdb` file extension.

-keyStorePassword
The password that protects the keystore. (String, required)

-keyStorePasswordVerify
The password that protects the keystore. (String, required)

Optional parameters

-keyStoreProvider
The provider used to implement the keystore. (String, optional)

-keyStoreIsFileBased
Set the value of this parameter to `true` if the keystore is file based. Set the value of this parameter to `false` for hardware crypto keystores. (Boolean, optional)

-keyStoreHostList
A list of host names that indicate from where the keystore is remotely managed, separated by commas. (String, optional)

-keyStoreInitAtStartup
Set the value of this parameter to `true` if the keystore is initialized at startup. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-keyStoreReadOnly
Set the value of this parameter to `true` if you cannot write to the keystore. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-keyStoreStashFile
Set the value of this parameter to `true` if you want to create stash files for CMS type keystore. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-enableCryptoOperations
Specifies if the keystore object will be used for hardware cryptographic operations or not. The default value is `false`. (Boolean, optional)

-keyStoreDescription
Specifies user defined text to describe the keystore of interest. (String, optional)

-keyStoreUsage
Specifies the keystore usage of interest. Specify `SSLKeys`, `KeySetKeys`, `RootKeys`, `DeletedKeys`, `DefaultSigners`, or `RSATokenKeys`. (String, optional)

-scopeName
The name that uniquely identifies the management scope, for example: `(cell):localhostNode01Cell`. (String, optional)

-controlRegionUser
Specifies the control region user to create a writable keystore object for the control regions key ring. Specify this option for SAF key rings when SAF writable key rings is enabled. (String, optional)

-servantRegionUser
Specifies the servant region user to create a writable keystore object for the servant regions key ring. Specify this option for SAF key rings when SAF writable key rings is enabled. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createKeyStore {-keyStoreName testKS -keyStoreType JCEKS
-keyStoreLocation c:/temp/testKeyFile.p12 -keyStorePassword testpwd
-keyStorePasswordVerify testpwd -keyStoreIsFileBased true -keyStoreInitAtStartup
true -keyStoreReadOnly false}
```

- Using Jython string:

```
AdminTask.createKeyStore('[-keyStoreName testKS -keyStoreType JCEKS -keyStoreLocation
c:/temp/testKeyFile.p12 -keyStorePassword testpwd -keyStorePasswordVerify testpwd
-keyStoreIsFileBased true -keyStoreInitAtStartup true -keyStoreReadOnly false]')
```

- Using Jython list:

```
AdminTask.createKeyStore(['-keyStoreName', 'testKS', '-keyStoreLocation', '-keyStoreType',
'JCEKS', 'c:/temp/testKeyFile.p12', '-keyStorePassword', 'testpwd',
'-keyStorePasswordVerify', 'testpwd', '-keyStoreIsFileBased', 'true',
'-keyStoreInitAtStartup', 'true', '-keyStoreReadOnly', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createKeyStore {-interactive}
```

- Using Jython:

```
AdminTask.createKeyStore('-interactive')
```

createCMSKeyStore

The **createCMSKeyStore** command creates a CMS keystore database and the keystore settings in the configuration.

Required parameters

-cmsKeyStoreURI
The URI of the CMS keystore. (String, required)

-pluginHostName
The host name of the plug-in. (String, required)

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createCMSKeyStore {-cmsKeyStoreURI CMSKeystoreURI -pluginHostName myHostName}
```

- Using Jython string:

```
AdminTask.createCMSKeyStore('-cmsKeyStoreURI CMSKeystoreURI -pluginHostName myHostName')
```

- Using Jython list:

```
AdminTask.createCMSKeyStore(['-cmsKeyStoreURI', 'CMSKeystoreURI', '-pluginHostName',
'myHostName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createCMSKeyStore {-interactive}
```

- Using Jython:

```
AdminTask.createCMSKeyStore('-interactive')
```

deleteKeyStore

The **deleteKeyStore** command deletes the settings of a keystore from the configuration and the keystore file.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore that you want to delete. (String, required)

Optional parameters

-scopeName
The name that uniquely identifies the management scope, for example: `(cell):localhostNode01Cell`. (String, optional)

-removeKeyStoreFile
Specifies whether to remove the keystore file. Specify `true` to remove the keystore file or `false` to keep the keystore file in your configuration. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyStore {-keyStoreName testKS}
```

- Using Jython string:

```
AdminTask.deleteKeyStore('[-keyStoreName testKS]')
```

- Using Jython list:

```
AdminTask.deleteKeyStore(['-keyStoreName', 'testKS'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyStore {-interactive}
```

- Using Jython:

```
AdminTask.deleteKeyStore('-interactive')
```

exchangeSigners

The **exchangeSigners** command exchange signer certificate between keystores.

Required parameters

-keyStoreName1
The name that uniquely identifies a keystore. You must specify a second keystore name using the keyStoreName2 parameter. (String, required)

-keyStoreName2
The name that uniquely identifies a keystore. You must specify a second keystore name using the keyStoreName1 parameter. (String, required)

Optional parameters

-keyStoreScope1
The scope name of the keystore that you specified with the keyStoreName1 parameter. (String, optional)

-keyStoreScope2
The scope name of the keystore that you specified with the keyStoreName2 parameter. (String, optional)

-certificateAlaisList1
A list of aliases separated by a comma. (String, optional)

-certificateAliasList2
A list of aliases separated by a comma. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask exchangeSigners {-keyStoreName1 testKS -certificateAliasList1 testCert1
-keyStoreName2 secondKS -certificateAlaisList2 certAlis}
```

- Using Jython string:

```
AdminTask.exchangeSigners('[-keyStoreName1 testKS -certificateAliasList1 testCert1
-keyStoreName2 secondKS -certificateAlaisList2 certAlis]')
```

- Using Jython list:

```
AdminTask.exchangeSigners(['-keyStoreName1', 'testKS', '-certificateAliasList1',
'testCert1', '-keyStoreName2', 'secondKS', '-certificateAlaisList2',
'certAlis'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask exchangeSigners {-interactive}
```

- Using Jython:

```
AdminTask.exchangeSigners('-interactive')
```

getKeyStoreInfo

The **getKeyStoreInfo** command displays the settings of a particular keystore.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore. (String, required)

Optional parameters

-scopeName
The name that uniquely identifies the management scope, for example: `(cell):localhostNode01Cell`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getKeyStoreInfo {-name testKS}
```

- Using Jython string:

```
AdminTask.getKeyStoreInfo('[-name testKS]')
```

- Using Jython list:

```
AdminTask.getKeyStoreInfo(['-name', 'testKS'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getKeyStoreInfo {-interactive}
```

- Using Jython:

```
AdminTask.getKeyStoreInfo('-interactive')
```

listKeyFileAliases

The **listKeyFileAliases** command lists the certificates in a keystore file.

Required parameters

-keyFilePath
The path of the key file. (String, required)

-keyFilePassword
The password for the key file. (String, required)

-keyFileType
The key file type. (String, required)

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeyFileAliases {-keyFilePath /temp/testKeyFile.p12
-keyFilePassword testPwd -keyFileType PKCS12}
```

- Using Jython string:

```
AdminTask.listKeyFileAliases('[-keyFilePaht /temp/testKeyFile.p12
-keyFilePassword testPwd -keyFileType PKCS12]')
```

- Using Jython list:

```
AdminTask.listKeyFileAliases(['-keyFilePaht', '/temp/testKeyFile.p12',
'-keyFilePassword', 'testPwd', '-keyFileType', 'PKCS12'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeyFileAliases {-interactive}
```

- Using Jython:

```
AdminTask.listKeyFileAliases('-interactive')
```

listKeyStores

The **listKeyStores** command lists the keystore for a particular scope.

Required parameters

None.

Optional parameters

-scopeName
Specifies the name that uniquely identifies the management scope, for example: `(cell):localhostNode01Cell`. (String, optional)

-all
Specify the value of this parameter as `true` to list all keystores. This parameter overrides the scopeName parameter. The default value is `false`. (Boolean, optional)

-keyStoreUsage
Specifies the keystore usage of interest. Specify `SSLKeys`, `KeySetKeys`, `RootKeys`, `DeletedKeys`, `DefaultSigners`, or `RSATokenKeys`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeyStores
```

- Using Jython:

```
AdminTask.listKeyStores()
```

Interactive mode example usage:

listKeyStoreTypes

The **listKeyStoreTypes** command lists all valid keystore types.

Required parameters

None.

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeyStoreTypes
```

- Using Jython:

```
AdminTask.listKeyStoreTypes()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeyStoreTypes {-interactive}
```

- Using Jython string:

```
AdminTask.listKeyStoreTypes('-interactive')
```

modifyKeyStore

The **modifyKeyStore** command modifies attributes for an existing keystore. Only some keystore attributes are modifiable, depending on what you are modifying. Use the following guidelines to use the command:

- To use this command to change the keystore file that the keystore object references, specify the keyStoreName, keyStoreLocation, keyStoreType, and keyStorePassword parameters.
-

Required parameters

-keyStoreName
Specifies the unique name that identifies the keystore. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the keystore. (String, optional)

-keyStoreProvider
Specifies the provider for the keystore. (String, optional)

-keyStoreType
Specifies one of the predefined keystore types. Valid values are JCEKS, CMSKS, PKCS12, PKCS11, and JKS. (String, optional)

-keyStoreLocation
Specifies the fully qualified location of the keystore file. To modify the location of the keystore file, you must specify the keyStoreLocation, keyStoreType, keyStorePassword, and keyStoreName parameters. (String, optional)

-keyStorePassword
Specifies the password to open the keystore. Use the changeKeystorePassword command to change the password of the keystore. (String, optional)

-keyStoreIsFileBased
Specifies whether the keystore is file based. To modify whether the keystore is file-based, specify the keyStoreIsFileBased and keyStoreName parameters. (Boolean, optional)

-keyStoreInitAtStartup
Specifies whether the keystore initiates at server startup. To modify whether the keystore initiates at server startup, specify the keyStoreInitAtStartup and keyStoreName parameters. (Boolean, optional)

-keyStoreReadOnly
Specifies whether the keystore is writable. To modify whether the keystore is read-only, specify the keyStoreReadOnly and keyStoreName parameters. (Boolean, optional)

-keyStoreDescription
Specifies a statement that describes the keystore. To modify the keystore description, specify the keyStoreDescription and keyStoreName parameters. (String, optional)

-keyStoreUsage
Specifies the keystore usage of interest. Specify SSLKeys, KeySetKeys, RootKeys, DeletedKeys, DefaultSigners, or RSATokenKeys. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifyKeyStore {-keyStoreName CellDefaultKeyStore
-keyStoreLocation c:/temp/testKeyFile.p12 -keyStoreType JCEKS
-keyStorePassword my1password}
```

- Using Jython:

```
AdminTask.modifyKeyStore('-keyStoreName CellDefaultKeyStore -keyStoreLocation
c:/temp/testKeyFile.p12 -keyStoreType JCEKS -keyStorePassword my1password')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyKeyStore {-interactive}
```

- Using Jython:

```
AdminTask.modifyKeyStore('-interactive')
```

SSLConfigCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the SSLConfigCommands group can be used to create and manage Secure Sockets Layer (SSL) configurations and properties.

The SSLConfigCommands command group for the AdminTask object includes the following commands:

createSSLConfig

The createSSLConfig command creates an SSL configuration that is based on key store and trust store settings. You can use the SSL configuration settings to make the SSL connections.

Target object

None.

Required parameters

-alias
The name of the alias. (String, required)

-trustStoreNames
The key store that holds trust information used to validate the trust from remote connections. (String, required)

-keyStoreName
The key store that holds the personal certificates that provide identity for the connection. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

-clientKeyAlias
The certificate alias name for the client. (String, optional)

-serverKeyAlias
The certificate alias name for the server. (String, optional)

-type
The type of SSL configuration. (String, optional)

-clientAuthentication
Set the value of this parameter to `true` to request client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-securityLevel
The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, required)

-enabledCiphers
A list of ciphers used during SSL handshake. (String, optional)

-jsseProvider
One of the JSSE providers. (String, optional)

-clientAuthenticationSupported
Set the value of this parameter to `true` to support client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-sslProtocol
The protocol type for the SSL handshake. Valid values include: `SSL_TLS`, `SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`. (String, optional)

-trustManagerObjectNames
A list of trust managers separated by commas. (String, optional)

-trustStoreScopeName
The management scope name of the trust store. (String, optional)

-keyStoreScopeName
The management scope name of the key store. (String, optional)

-keyManagerName
- Specifies the name of the Key Manager. (String, optional)

-keyManagerScopeName
Specifies the scope of the key manager. (String, optional)

-ssslKeyRingName
Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional)

-v3timeout
- Specifies the time out in seconds for System SSL configuration types. Values range from 1 to 86400. (String, optional)

Example output

The command returns the configuration object name of the new SSL configuration object.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask createSSLConfig {-alias testSSLCfg  -clientKeyAlias key1
-serverKeyAlias key2 -trustStoreNames trustKS –keyStoreName
testKS -keyManagerName testKeyMgr}
```

- Using Jython string:

```
AdminTask.createSSLConfig('[-alias testSSLCfg  -clientKeyAlias key1
-serverKeyAlias key2 -trustStoreNames trustKS –keyStoreName
testKS -keyManagerName testKeyMgr]')
```

- Using Jython list:

```
AdminTask.createSSLConfig(['-alias', 'testSSLCfg', '-clientKeyAlias',
'key1', '-serverKeyAlias', 'key2', '-trustStoreNames', 'trustKS',
'–keyStoreName', 'testKS', '-keyManagerName', 'testKeyMgr'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createSSLConfig {-interactive}
```

- Using Jython:

```
AdminTask.createSSLConfig('-interactive')
```

createSSLConfigProperty

The createSSLConfigProperty command creates a property for an SSL configuration. Use this command to set SSL configuration settings that are different than the settings in the SSL configuration object.

Target object

None.

Required parameters

-sslConfigAliasName
The alias name of the SSL configuration. (String, required)

-propertyName
The name of the property. (String, required)

-propertyValue
The value of the property. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

Example output

The command does not return output.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask createSSLConfigProperty {-sslConfigAliasName NodeDefaultSSLSettings
-scopeName (cell):localhostNode01Cell:(node):localhostNode01 -propertyName
test.property -propertyValue testValue}
```

- Using Jython string:

```
AdminTask.createSSLConfigProperty('[-sslConfigAliasName NodeDefaultSSLSettings
-scopeName (cell):localhostNode01Cell:(node):localhostNode01 -propertyName
test.property -propertyValue testValue]')
```

- Using Jython list:

```
AdminTask.createSSLConfigProperty(['-sslConfigAliasName', 'NodeDefaultSSLSettings',
'-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01', '-propertyName',
'test.property', '-propertyValue', 'testValue'])
```

Examples:

Batch mode example usage:

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createSSLConfigProperty {-interactive}
```

- Using Jython:

```
AdminTask.createSSLConfigProperty('-interactive')
```

deleteSSLConfig

The deleteSSLConfig command deletes the SSL configuration object that you specify from the configuration.

Target object

None.

Required parameters and return values

-alias
The name of the alias. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

Example output

The command does not return output.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteSSLConfig {-alias NodeDefaultSSLSettings -scopeName
(cell):localhostNode01Cell:(node):localhostNode01}
```

- Using Jython string:

```
AdminTask.deleteSSLConfig('[-alias NodeDefaultSSLSettings -scopeName
(cell):localhostNode01Cell:(node):localhostNode01]')
```

- Using Jython list:

```
AdminTask.deleteSSLConfig(['-alias', 'NodeDefaultSSLSettings', '-scopeName',
'(cell):localhostNode01Cell:(node):localhostNode01'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteSSLConfig {-interactive}
```

- Using Jython:

```
AdminTask.deleteSSLConfig('-interactive')
```

getInheritedSSLConfig

The getInheritedSSLConfig command returns the SSL configuration alias and certificate alias from which a given management scope and direction inherits its SSL configuration information. This command only returns inheritance information; it does not return information about an SSL configuration that is effective for a give scope.

For example, by default in a Network Deployment environment, there are different SSL configuration effective at the cell and node levels. If you issue the getInheritedSSLConfig command, specifying the nodes management scope, you get the name of the SSL configuration for the cell, not the effective SSL configuration of the node, because the node inherits its configuration information from the cell.

Target object

None.

Required parameters and return values

-scopeName
The name of the management scope for which you want to find out where that management schope will inherit its SSL configuration. (String, required)

Optional parameters

None.

Example output

The command returns the SSL configuration alias and certificate alias from which the specified management scope and direction inherits its SSL configuration information.

Examples:

- Using Jacl:

```
$AdminTask getInheritedSSLConfig {-scopeName (cell):localhostNode01Cell:(node):localhostNode01 -direction inbound}
CellDefaultSSLSettings,null
```

- Using Jython string:

```
AdminTask.getInheritedSSLConfig('[-scopeName
(cell):localhostNode01Cell:(node):localhostNode01 -direction inbound]')
CellDefaultSSLSettings,null
```

getSSLConfig

The getSSLConfig command obtains information about an SSL configuration and displays the settings.

Target object

None.

Required parameters and return values

-alias
The name of the alias. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

Example output:

The command returns information about the SSL configuration of interest.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfig {-alias NodeDefaultSSLSettings -scopeName
(cell):localhostNode01Cell:(node):localhostNode01}
```

- Using Jython string:

```
AdminTask.getSSLConfig('[-alias NodeDefaultSSLSettings -scopeName
(cell):localhostNode01Cell:(node):localhostNode01]')
```

- Using Jython list:

```
AdminTask.getSSLConfig(['-alias', 'NodeDefaultSSLSettings', '-scopeName',
'(cell):localhostNode01Cell:(node):localhostNode01'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfig {-interactive}
```

- Using Jython:

```
AdminTask.getSSLConfig('-interactive')
```

getSSLConfigProperties

The getSSLConfigProperties command obtains information about SSL configuration properties.

Target object

None.

Required parameters and return values

-alias
The name of the alias. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

Example output

The command returns additional information about the SSL configuration properties.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfigProperties {-sslConfigAliasName NodeDefaultSSLSettings
-scopeName (cell):localhostNode01Cell:(node):localhostNode01}
```

- Using Jython string:

```
AdminTask.getSSLConfigProperties('[-sslConfigAliasName NodeDefaultSSLSettings
-scopeName (cell):localhostNode01Cell:(node):localhostNode01]')
```

- Using Jython list:

```
AdminTask.getSSLConfigProperties(['-sslConfigAliasName', 'NodeDefaultSSLSettings',
'-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfigProperties {-interactive}
```

- Using Jython:

```
AdminTask.getSSLConfigProperties('-interactive')
```

listSSLCiphers

The listSSLCiphers command lists the SSL ciphers.

Target object

None.

Required parameters

-securityLevel
The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, required)

Optional parameters

-sslConfigAliasName
The alias name of the SSL configuration. (String, optional)

-scopeName
The name of the scope. (String, optional)

Example output

The command returns a list of SSL ciphers.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask listSSLCiphers {-sslConfigAliasName testSSLCfg
-securityLevel HIGH}
```

- Using Jython string:

```
AdminTask.listSSLCiphers('[-sslConfigAliasName testSSLCfg
-securityLevel HIGH]')
```

- Using Jython list:

```
AdminTask.listSSLCiphers(['-sslConfigAliasName', 'testSSLCfg',
'-securityLevel', 'HIGH'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSSLCiphers {-interactive}
```

- Using Jython:

```
AdminTask.listSSLCiphers('-interactive')
```

listSSLConfigs

The listSSLConfigs command lists the defined SSL configurations within a management scope.

Target object

None.

Optional parameters

-scopeName
The name of the scope. (String, optional)

-displayObjectName
Set the value of this parameter to `true` to list the SSL configuration objects within the scope. Set the value of this parameter to `false` to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all SSL configurations. This parameter overrides the scopeName parameter. The default value is `false`. (Boolean, optional)

Example output

The command returns a list of defined SSL configurations.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigs {-scopeName (cell): localhostNode01Cell:(node):localhostNode01
-displayObjectName true}
```

- Using Jython string:

```
AdminTask.listSSLConfigs('[-scopeName (cell):localhostNode01Cell:(node):localhostNode01
-displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listSSLConfigs(['-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01',
'-displayObjectName', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigs {-interactive}
```

- Using Jython:

```
AdminTask.listSSLConfigs('-interactive')
```

listSSLConfigProperties

The listSSLConfigProperties command lists the properties for a SSL configuration.

Target object

None.

Required parameters

-alias
The alias name of the SSL configuration. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

-displayObjectName
Set the value of this parameter to `true` to list the SSL configuration objects within the scope. Set the value of this parameter to `false` to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional)

Example output

The command returns SSL configuration properties.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigProperty {-alias SSL123 -scopeName
(cell):localhostNode01Cell:(node):localhostNode01 -displayObjectName true}
```

- Using Jython string:

```
AdminTask.listSSLConfigProperty('[-alias SSL123 -scopeName
(cell):localhostNode01Cell:(node):localhostNode01 -displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listSSLConfigProperty(['-alias', 'SSL123', '-scopeName',
'(cell):localhostNode01Cell:(node):localhostNode01', '-displayObjectName', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigProperties {-interactive}
```

- Using Jython:

```
AdminTask.listSSLConfigProperties('-interactive')
```

listSSLRepertoires

The listSSLRepertoires command lists all of the Secure Sockets Layer (SSL) configuration instances that you can associate with an SSL inbound channel.If you create a new SSL alias using the administrative console, the alias name is automatically created in the *node_name/alias_name* format. However, if you create a new SSL alias using the wsadmin tool, you must create the SSL alias and specify both the node name and alias name in the *node_name/alias_name* format.

Target object

SSLInboundChannel instance for which the SSLConfig candidates are listed.

Required parameters

None.

Optional parameters

None.

Sample output

The command returns a list of eligible SSL configuration object names.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask listSSLRepertoires SSL_3(cells/mybuildCell01/nodes/mybuildNode01/servers/
server2|server.xml#SSLInboundChannel_1093445762330)
```

- Using Jython string:

```
print AdminTask.listSSLRepertoires('SSL_3(cells/mybuildCell01/nodes/mybuildNode01/
servers/server2|server.xml#SSLInboundChannel_1093445762330)')
```

- Using Jython list:

```
print AdminTask.listSSLRepertoires('SSL_3(cells/mybuildCell01/nodes/mybuildNode01/
servers/server2|server.xml#SSLInboundChannel_1093445762330)')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSSLRepertoires {-interactive}
```

- Using Jython:

```
print AdminTask.listSSLRepertoires('-interactive')
```

modifySSLConfig

The modifySSLConfig command modifies the settings of an existing SSL configuration.

Target object

None.

Required parameters

-alias
The name of the alias. (String, required)

Optional parameters

-scopeName
The name of the scope. (String, optional)

-clientKeyAlias
The certificate alias name for the client. (String, optional)

-serverKeyAlias
The certificate alias name for the server. (String, optional)

-clientAuthentication
Set the value of this parameter to `true` to request client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-securityLevel
The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, required)

-enabledCiphers
A list of ciphers used during SSL handshake. (String, optional)

-jsseProvider
One of the JSSE providers. (String, optional)

-clientAuthenticationSupported
Set the value of this parameter to `true` to support client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-sslProtocol
The protocol type for the SSL handshake. Valid values include: `SSL_TLS`, `SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`. (String, optional)

-trustManagerObjectNames
A list of trust managers separated by commas. (String, optional)

-trustStoreName
The key store that holds trust information used to validate the trust from remote connections. (String, optional)

-trustStoreScopeName
The management scope name of the trust store. (String, optional)

-keyStoreName
The key store that holds the personal certificates that provide identity for the connection. (String, optional)

-keyStoreScopeName
The management scope name of the key store. (String, optional)

-keyManagerName
- Specifies the name of the Key Manager. (String, optional)

-keyManagerScopeName
Specifies the scope of the key manager. (String, optional)

-ssslKeyRingName
Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional)

-v3timeout
- Specifies the time out in seconds for System SSL configuration types. Values range from 1 to 86400. (String, optional)

Example output

The command does not return output.

Examples:

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifySSLConfig {-alias testSSLCfg -clientKeyAlias tstKey1
-serverKeyAlias tstKey2 -securityLevel LOW}
```

- Using Jython string:

```
AdminTask.modifySSLConfig('[-alias testSSLCfg -clientKeyAlias tstKey1
-serverKeyAlias tstKey2 -securityLevel LOW]')
```

- Using Jython list:

```
AdminTask.modifySSLConfig(['-alias', 'testSSLCfg', '-clientKeyAlias', 'tstKey1',
'-serverKeyAlias', 'tstKey2', '-securityLevel', 'LOW'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifySSLConfig {-interactive}
```

- Using Jython:

```
AdminTask.modifySSLConfig('-interactive')
```

SSLConfigGroupCommands group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the SSLConfigGroupCommands group can be used to create and manage SSL configuration groups.

The SSLConfigGroupCommands command group for the AdminTask object includes the following commands:

- "deleteSSLConfigGroup"
- "getSSLConfigGroup" on page 221
- "listSSLConfigGroups" on page 222
- "modifySSLConfigGroup" on page 223

deleteSSLConfigGroup

The deleteSSLConfigGroup command deletes a SSL configuration group from the configuration.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the SSL configuration group. (String, required)

-direction
Specifies the direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required)

Optional parameters

-scopeName
Specifies the name that uniquely identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteSSLConfigGroup {-name createSSLCfgGrp -direction inbound}
```

- Using Jython string:

```
AdminTask.deleteSSLConfigGroup('[-name createSSLCfgGrp -direction inbound]')
```

- Using Jython list:

```
AdminTask.deleteSSLConfigGroup(['-name', 'createSSLCfgGrp', '-direction', 'inbound'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteSSLConfigGroup {-interactive}
```

- Using Jython:

```
AdminTask.deleteSSLConfigGroup('-interactive')
```

getSSLConfigGroup

The getSSLConfigGroup command returns information about a SSL configuration setting.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the SSL configuration group. (String, required)

-direction
Specifies the direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required)

Optional parameters

-scopeName
Specifies the name that uniquely identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfigGroup {-name createSSLCfgGrp -direction inbound}
```

- Using Jython string:

```
AdminTask.getSSLConfigGroup('[-name createSSLCfgGrp -direction inbound]')
```

- Using Jython list:

```
AdminTask.getSSLConfigGroup(['-name', 'createSSLCfgGrp', '-direction', 'inbound'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getSSLConfigGroup {-interactive}
```

- Using Jython:

```
AdminTask.getSSLConfigGroup('-interactive')
```

listSSLConfigGroups

The listSSLConfigGroups command lists the SSL configuration groups within a scope and a direction.

Target object

None.

Required parameters

None.

Optional parameters

-direction
Specifies the direction to which the SSL configuration applies. Valid values include `inbound` or `outbound`. (String, optional)

-scopeName
Specifies the name that uniquely identifies the management scope. (String, optional)

-displayObjectName
If you set this parameter to true, the command returns a list of all of the SSL configuration group objects within the scope. If you set this parameter to `false`, the command returns a list of strings that contain the SSL configuration name and management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all SSL configuration groups. This parameter overrides the scopeName parameter. The default value is `false`. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigGroups {-displayObjectName true}
```

- Using Jython string:

```
AdminTask.listSSLConfigGroups('[-displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listSSLConfigGroups(['-displayObjectName' 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSSLConfigGroups {-interactive}
```

- Using Jython:

```
AdminTask.listSSLConfigGroups('-interactive')
```

modifySSLConfigGroup

The modifySSLConfigGroup command modifies the setting of an existing SSL configuration group.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the SSL configuration group. (String, required)

-direction
Specifies the direction to which the SSL configuration applies. Valid values include `inbound` or `outbound`. (String, required)

Optional parameters

-certificateAlias
Specifies a unique name to identify a certificate. (String, optional)

-scopeName
Specifies the name that uniquely identifies the management scope. (String, optional)

-sslConfigAliasName
Specifies the alias that uniquely identifies the SSL configurations in the group. (String, optional)

-sslConfigScopeName
Specifies the scope that uniquely identifies the SSL configurations in the group. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifySSLConfigGroup {-name createSSLCfgGrp -direction inbound -certificateAlias alias2}
```

- Using Jython string:

```
AdminTask.modifySSLConfigGroup('[-name createSSLCfgGrp -direction inbound -certificateAlias alias2]')
```

- Using Jython list:

```
AdminTask.modifySSLConfigGroup(['-name', 'createSSLCfgGrp', '-direction', 'inbound', '-certificateAlias', 'alias2'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifySSLConfigGroup {-interactive}
```

- Using Jython:

```
AdminTask.modifySSLConfigGroup('-interactive')
```

TrustManagerCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the TrustManagerCommands group can be used to create, delete, and query trust manager settings in your configuration. You can also use these commands to create a custom trust manager for a pure client.

The TrustManagerCommands command group for the AdminTask object includes the following commands:

- "createTrustManager" on page 224
- "deleteTrustManager" on page 224
- "getTrustManager" on page 225

createTrustManager

The **createTrustManager** command creates a trust manager in the configuration.

Target object

None

Parameters and return values

-name
The name that uniquely identifies the trust manager. (String, required)

-scopeName
The name of the scope. (String, optional)

-provider
Specifies the provider. (String, optional)

-algorithm
Specifies the algorithm name of the trust manager or key manager. (String, optional)

-trustManagerClass
Specifies the custom class that implements the javax.net.s sl.TrustManager interface. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:
```
$AdminTask createTrustManager {-name testTM}
```
- Using Jython string:
```
AdminTask.createTrustManager ('[-name testTM]')
```
- Using Jython list:
```
AdminTask.createTrustManager (['-name', 'testTM'])
```

Interactive mode example usage:

- Using Jacl:
```
$AdminTask createTrustManager {-interactive}
```
- Using Jython string:
```
AdminTask.deleteTrustManager ('[-interactive]')
```
- Using Jython list:
```
AdminTask.createTrustManager (['-interactive'])
```

deleteTrustManager

The **deleteTrustManager** command deletes the trust manager settings from the configuration.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the trust manager. (String, required)

Optional parameters

-scopeName
Specifies the name of the scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask deleteTrustManager {-name testTM}
  ```
- Using Jython string:
  ```
  AdminTask.deleteTrustManager('[-name testTM]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteTrustManager(['-name', 'testTM'])
  ```

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask deleteTrustManager {-interactive}
  ```
- Using Jython:
  ```
  AdminTask.deleteTrustManager('-interactive')
  ```

getTrustManager

The **getTrustManager** command obtains the setting of a trust manager.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the trust manager. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask getTrustManager {-name testTM}
  ```
- Using Jython string:
  ```
  AdminTask.getTrustManager('[-name testTM]')
  ```
- Using Jython list:
  ```
  AdminTask.getTrustManager(['-name', 'testTM'])
  ```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getTrustManager {-interactive}
```

- Using Jython:

```
AdminTask.getTrustManager('-interactive')
```

listTrustManagers

The **listTrustManagers** command lists the trust managers within a particular management scope.

Target object

None.

Required parameters

None.

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

-displayObjectName
Set the value of this parameter to true to list the trust manager objects within a scope. Set the value of this parameter to false to list the strings that contain the trust manager name and management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all trust managers. This parameter overrides the scopeName parameter. The default value is false. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listTrustManagers {-displayObjectName true}
```

- Using Jython string:

```
AdminTask.listTrustManagers('[-displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listTrustManagers(['-displayObjectName', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listTrustManagers {-interactive}
```

- Using Jython:

```
AdminTask.listTrustManagers('-interactive')
```

modifyTrustManager

The **modifyTrustManager** command changes existing trust manager settings.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the trust manager. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

-provider
Specifies the provider name of the trust manager. (String, optional)

-algorithm
Specifies the algorithm name of the trust manager. (String, optional)

-trustManagerClass
Specifies a class that implements the `javax.net. sslX509Tr ustManager` interface. You cannot use this parameter with the provider or algorithm parameters. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask modifyTrustManager {-name testTM -trustManagerClass test.trust.manager}
  ```
- Using Jython string:
  ```
  AdminTask.modifyTrustManager('[-name testTM -trustManagerClass test.trust.manager]')
  ```
- Using Jython list:
  ```
  AdminTask.modifyTrustManager(['-name', 'testTM', '-trustManagerClass', 'test.trust.manager'])
  ```

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask modifyTrustManager {-interactive}
  ```
- Using Jython:
  ```
  AdminTask.modifyTrustManager('-interactive')
  ```

KeySetCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the KeySetCommands group can be used to create, delete, and query for key set settings in your configuration.

The KeySetCommands command group for the AdminTask object includes the following commands:

- "createKeySet"
- "deleteKeySet" on page 229
- "generateKeyForKeySet" on page 229
- "getKeySet" on page 230
- "listKeySets" on page 231
- "modifyKeySet" on page 232

createKeySet

The **createKeySet** command creates the key set settings in the configuration. Use this command to control key instances that have the same type.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set. (String, required)

-aliasPrefix
Specifies the prefix for the key alias when a new key generates. (String, required)

-password
Specifies the password that protects the key in the keystore. (String, required)

-maxKeyReferences
Specifies the maximum number of key references from the returned keys in the key set of interest. (Integer, required)

-keyStoreName
Specifies the keystore that contains the keys. (String, required)

Optional parameters

-scopeName
Specifies the unique name of the management scope. (String, optional)

-deleteOldKeys
Set the value of this parameter to `true` to delete old keys when new keys are generated. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-keyGenerationClass
Specifies the class that is used to generate new keys in the key set. (String, optional)

-keyStoreScopeName
Specifies the management scope where the keystore is located. (String, optional)

-isKeyPair
Set the value of this parameter to `true` if the keys in the key set are key pairs. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

Example output

The command returns the configuration object name of the key set object that you created.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createKeySet {-name testKeySet –aliasPrefix test –password pwd
–maxKeyReferences 2 –deleteOldKeys true –keyStoreName testKeyStore –isKeyPair false}
```

- Using Jython string:

```
AdminTask.createKeySet('[-name testKeySet –aliasPrefix test –password pwd
–maxKeyReferences 2 –deleteOldKeys true –keyStoreName testKeyStore –isKeyPair false]')
```

- Using Jython list:

```
AdminTask.createKeySet(['-name', 'testKeySet', '–aliasPrefix', 'test',
'–password', 'pwd', '–maxKeyReferences', '2', '–deleteOldKeys', 'true',
'–keyStoreName', 'testKeyStore', '–isKeyPair', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createKeySet {-interactive}
```

- Using Jython string:
  ```
  AdminTask.createKeySet ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.createKeySet (['-interactive'])
  ```

deleteKeySet

The **deleteKeySet** command deletes the settings of a key set from the configuration.

Target object

None.

Required parameters

-name
The name that uniquely identifies the key set. (String, required)

Optional parameters

-scopeName
Specifies the unique name of the management scope. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:
- Using Jacl:
  ```
  $AdminTask deleteKeySet{ -name testKeySet}
  ```
- Using Jython string:
  ```
  AdminTask.deleteKeySet('[-name testKeySet]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteKeySet(['-name', 'testKeySet'])
  ```

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask deleteKeySet {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.deleteKeySet ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteKeySet (['-interactive'])
  ```

generateKeyForKeySet

The **generateKeyForKeySet** command generates keys for the keys in the key set.

Target object

None.

Required parameters

-keySetName
Specifies the name of the key set. (String, required)

Optional parameters

-keySetScope
Specifies the scope of the key set. (String, optional)

-keySetSaveConfig
Set the value of this parameter to `true` to save the configuration of the key set. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:
  `$AdminTask generateKeyForKeySet{ -keySetName testKeySet }`
- Using Jython string:
  `AdminTask.generateKeyForKeySet('[-keySetName testKeySet]')`
- Using Jython list:
  `AdminTask.generateKeyForKeySet(['-keySetName', 'testKeySet'])`

Interactive mode example usage:

- Using Jacl:
  `$AdminTask generateKeyForKeySet {-interactive}`
- Using Jython string:
  `AdminTask.generateKeyForKeySet ('[-interactive]')`
- Using Jython list:
  `AdminTask.generateKeyForKeySet (['-interactive'])`

getKeySet

The **getKeySet** command displays the settings of a particular key set.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set. (String, required)

Optional parameters

-scopeName
Specifies the unique name of the management scope. (String, optional)

Example output

The command returns the settings of the specified key set group.

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask getKeySet {-name testKeySet}
  ```
- Using Jython string:
  ```
  AdminTask.getKeySet ('[-name testKeySet]')
  ```
- Using Jython list:
  ```
  AdminTask.getKeySet (['-name', 'testKeySet'])
  ```

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask getKeySet {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getKeySet ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getKeySet (['-interactive'])
  ```

listKeySets

The **listKeySets** command lists the key sets in a particular scope.

Target object

None.

Required parameters

None.

Optional parameters

-scopeName
Specifies the unique name of the management scope. (String, optional)

-displayObjectName
Set the value of this parameter to `true` to list the key set configuration objects within the scope. Set the value of this parameter to `false` if you want to list the strings that contain the key set group name and management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all key sets. This parameter overrides the scopeName parameter. The default value is `false`. (Boolean, optional)

Example output

The command returns the key sets for the scope that you specified.

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask listKeySets {-displayObjectName true}
  ```
- Using Jython string:

```
AdminTask.listKeySets ('[-displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listKeySets (['-displayObjectName', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeySets {-interactive}
```

- Using Jython string:

```
AdminTask.listKeySets ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listKeySets (['-interactive'])
```

modifyKeySet

The **modifyKeySet** command changes the settings of an existing key set.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set. (String, required)

Optional parameters

-scopeName
Specifies the unique name of the management scope. (String, optional)

-aliasPrefix
Specifies the prefix for the key alias when a new key generates. (String, optional)

-password
Specifies the password that protects the key in the keystore. (String, optional)

-maxKeyReferences
Specifies the maximum number of key references from the returned keys in the key set of interest. (Integer, optional)

-deleteOldKeys
Set the value of this parameter to `true` to delete old keys when new keys are generated. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-keyGenerationClass
Specifies the class that is used to generate new keys in the key set. (String, optional)

-keyStoreName
Specifies the keystore that contains the keys. (String, optional)

-keyStoreScopeName
Specifies the management scope where the keystore is located. (String, optional)

-isKeyPair
Set the value of this parameter to `true` if the keys in the key set are key pairs. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifyKeySet {-name testKeySet -maxKeyReferences 3
–deleteOldKeys false}
```

- Using Jython string:

```
AdminTask.modifyKeySet ('[-name testKeySet -maxKeyReferences 3
–deleteOldKeys false]')
```

- Using Jython list:

```
AdminTask.modifyKeySet (['-name', 'testKeySet', '-maxKeyReferences', '3',
'–deleteOldKeys', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyKeySet {-interactive}
```

- Using Jython string:

```
AdminTask.modifyKeySet ('[-interactive]')
```

- Using Jython list:

```
AdminTask.modifyKeySet (['-interactive'])
```

KeyReferenceCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the KeyReferenceCommands group can be used to create and manage the key reference settings for key set objects in your configuration.

The KeyReferenceCommands command group for the AdminTask object includes the following commands:

- "createKeyReference"
- "deleteKeyReference" on page 234
- "getKeyReference" on page 235
- "listKeyReferences" on page 236

createKeyReference

The **createKeyReference** command creates the key reference setting in the configuration for key set objects.

Target object

None.

Parameters and return values

-keySetName
The name that uniquely identifies the key set to which the key reference belongs. (String, required)

-keySetScope
The management scope of the key set. (String, optional)

-keyAlias
The alias name that identifies the key for the key set that you specify. (String, required)

-keyPassword
The password used for encrypting the key. (String, optional)

-keyPasswordVerify
The password used for encrypting the key. (String, optional)

-version
The version of the key reference. (String, optional)

-keyReferenceSaveConfig
Set the value of this parameter to `true` to save the key reference to the configuration. Otherwise, set the value to `false`. (String, optional)

- Returns: The configuration object name of the key reference scope object that you created.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createKeyReference {-keySetName testKeySet  -keyAlias testKey
–password testPWD –passwordVerify testPWD –keyReferenceSaveConfig true}
```

- Using Jython string:

```
AdminTask.createKeyReference ('[-keySetName testKeySet  -keyAlias testKey
–password testPWD –passwordVerify testPWD –keyReferenceSaveConfig true]')
```

- Using Jython list:

```
AdminTask.createKeyReference (['-keySetName', 'testKeySet', '-keyAlias', 'testKey',
'–password', 'testPWD', '–passwordVerify', 'testPWD', '–keyReferenceSaveConfig', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createKeyReference {-interactive}
```

- Using Jython string:

```
AdminTask.createKeyReference ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createKeyReference (['-interactive'])
```

deleteKeyReference

The **deleteKeyReference** command deletes a key reference object from the key set object in the configuration.

Target object

None.

Parameters and return values

-keySetName
The name that uniquely identifies the key set to which the key reference belongs. (String, required)

-keySetScope
The management scope of the key set. (String, optional)

-keyAlias
The alias name that identifies the key for the key set that you specify. (String, required)

- Returns: None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyReference { -keySetName testKeySet –keyAlias testKey }
```

- Using Jython string:

```
AdminTask.deleteKeyReference ('[-keySetName testKeySet –keyAlias testKey]')
```

- Using Jython list:

```
AdminTask.deleteKeyReference (['-keySetName', 'testKeySet', '–keyAlias', 'testKey'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteKeyReference {-interactive}
```

- Using Jython string:

```
AdminTask.deleteKeyReference ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteKeyReference (['-interactive'])
```

getKeyReference

The **getKeyReference** command displays the setting of a key reference object.

Target object

None.

Parameters and return values

-keySetName
The name that uniquely identifies the key set to which the key reference belongs. (String, required)

-keySetScope
The management scope of the key set. (String, optional)

-keyAlias
The alias name that identifies the key for the key set that you specify. (String, required)

- Returns: The settings of the key reference object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getKeyReference { -keySetName testKeySet –keyAlias testKey }
```

- Using Jython string:

```
AdminTask.getKeyReference ('[-keySetName testKeySet –keyAlias testKey]')
```

- Using Jython list:

```
AdminTask.getKeyReference (['-keySetName', 'testKeySet', '–keyAlias', 'testKey'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getKeyReference {-interactive}
```

- Using Jython string:

```
AdminTask.getKeyReference ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getKeyReference (['-interactive'])
```

listKeyReferences

The **listKeyReferences** command lists the key references for a particular key set in the configuration.

Target object

None.

Parameters and return values

-keySetName
The name that uniquely identifies the key set to which the key reference belongs. (String, required)

-keySetScope
The management scope of the key set. (String, optional)

- Returns: The configuration object name of the key reference scope object that you created.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeyRefereces { -keySetName testKeySet}
```

- Using Jython string:

```
AdminTask.listKeyRefereces ('[-keySetName testKeySet]')
```

- Using Jython list:

```
AdminTask.listKeyRefereces (['-keySetName', 'testKeySet'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeyReferences {-interactive}
```

- Using Jython string:

```
AdminTask.listKeyReferences ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listKeyReferences (['-interactive'])
```

KeySetGroupCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the KeySetGroupCommands group can be used to create and manage key set groups. Use these commands to manage groups of public, private, and shared keys.

The KeySetGroupCommands command group for the AdminTask object includes the following commands:

deleteKeySetGroup

The **deleteKeySetGroup** command deletes the settings of a key set group from the configuration.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set group. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

To list the valid management scopes, you can run the listManagementScope task. For example:

```
wsadmin>$AdminTask listManagementScopes
"scopeName: (cell):IBM-2143376CB9ECell03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ECellManager03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode02 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode04 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode05 "
```

Examples

Batch mode example usage:

- Using Jacl:
  ```
  $AdminTask deleteKeySetGroup {-name keySetGrp }
  ```
- Using Jython string:
  ```
  AdminTask.deleteKeySetGroup ('[-name keySetGrp]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteKeySetGroup (['-name', 'keySetGrp'])
  ```

Interactive mode example usage:

- Using Jacl:
  ```
  $AdminTask deleteKeySetGroup {-interactive}
  ```
- Using Jython:
  ```
  AdminTask.deleteKeySetGroup('-interactive')
  ```

generateKeyForKeySetGroup

The **generateKeysForKeySetGroup** command generates keys for all of the keys in the key sets that make up the key set group.

Target object

None.

Required parameters

-keySetGroupName
Specifies the name of the key set group. (String, required)

Optional parameters

-keySetGroupScope
Specifies the scope of the key set group. (String, optional)

To list the valid management scopes, you can run the listManagementScope task. For example:

```
wsadmin>$AdminTask listManagementScopes
"scopeName: (cell):IBM-2143376CB9ECell03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ECellManager03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode02 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode04 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode05 "
```

-keySetGroupUpdateRuntime
Specifies to update the environment to use the newly generated keys at run time. (Boolean, optional)

-keySetGroupSaveConfig
Specifies to automatically save the change to the security configuration. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

  ```
  $AdminTask generateKeyForKeySetGroup {-keySetGroupName keySetGrp}
  ```

- Using Jython string:

  ```
  AdminTask.generateKeyForKeySetGroup ('[-keySetGroupName keySetGrp]')
  ```

- Using Jython list:

  ```
  AdminTask.generateKeyForKeySetGroup (['-keySetGroupName', 'keySetGrp'])
  ```

Interactive mode example usage:

- Using Jacl:

  ```
  $AdminTask generateKeyForKeySetGroup {-interactive}
  ```

- Using Jython:

  ```
  AdminTask.generateKeyForKeySetGroup('-interactive')
  ```

getKeySetGroup

The **getKeySetGroup** command displays the settings of a particular key set group.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set group. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

To list the valid management scopes, you can run the listManagementScope task. For example:

```
wsadmin>$AdminTask listManagementScopes
"scopeName: (cell):IBM-2143376CB9ECell03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ECellManager03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode02 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode04 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode05 "
```

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getKeySetGroup { -name keySetGrp }
```

- Using Jython string:

```
AdminTask.getKeySetGroup ('[-name keySetGrp]')
```

- Using Jython list:

```
AdminTask.getKeySetGroup (['-name', 'keySetGrp'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getKeySetGroup {-interactive}
```

- Using Jython:

```
AdminTask.getKeySetGroup('-interactive')
```

listKeySetGroups

The **listKeySetGroups** command lists the key set groups for a particular scope.

Target object

None.

Required parameters

None.

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

To list the valid management scopes, you can run the listManagementScope task. For example:

```
wsadmin>$AdminTask listManagementScopes
"scopeName: (cell):IBM-2143376CB9ECell03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ECellManager03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode02 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode04 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode05 "
```

-displayObjectNames
If you set the value of this parameter to true, the command returns a list of all of the key set group objects within a scope. If you set the value of this parameter to false, the command returns a list of strings that contain the key set group name and management scope. (Boolean, optional)

-all
Specify the value of this parameter as `true` to list all key set groups. This parameter overrides the scopeName parameter. The default value is `false`. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeySetGroups {-displayObjectName true}
```

- Using Jython string:

```
AdminTask.listKeySetGroups ('[-displayObjectName true]')
```

- Using Jython list:

```
AdminTask.listKeySetGroups (['-displayObjectName', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listKeySetGroups {-interactive}
```

- Using Jython:

```
AdminTask.listKeySetGroups('-interactive')
```

modifyKeySetGroup

The **modifyKeySetGroup** command changes the settings of an existing key set group.

Target object

None.

Required parameters

-name
Specifies the name that uniquely identifies the key set group. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

To list the valid management scopes, you can run the listManagementScope task. For example:

```
wsadmin>$AdminTask listManagementScopes
"scopeName: (cell):IBM-2143376CB9ECell03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ECellManager03 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode02 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode04 "
"scopeName: (cell):IBM-2143376CB9ECell03:(node):IBM-2143376CB9ENode05 "
```

-autoGenerate
Set the value of this parameter to true if you want to automatically generate keys. If not, set the value to false. (Boolean, optional)

-wsScheduleName
Specifies the name of the scheduler to use to perform key generation. (String, optional)

-keySetObjectNames
A list of key set configuration names separated by colons (:). (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifyKeySetGroup {-name keySetGrp -autoGenerate false}
```

- Using Jython string:

```
AdminTask.modifyKeySetGroup ('[-name keySetGrp -autoGenerate false]')
```

- Using Jython list:

```
AdminTask.modifyKeySetGroup (['-name', 'keySetGrp', '-autoGenerate', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyKeySetGroup {-interactive}
```

- Using Jython:

```
AdminTask.modifyKeySetGroup('-interactive')
```

DynamicSSLConfigSelections command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the DynamicSSLConfigSelections group can be used to create, delete, and query dynamic SSL configuration selection objects.

The DynamicSSLConfigSelections command group for the AdminTask object includes the following commands:

- "deleteDynamicSSLConfigSelection"
- "getDynamicSSLConfigSelection"
- "listDynamicSSLConfigSelections" on page 242

deleteDynamicSSLConfigSelection

The **deleteDynamicSSLConfigSelection** command deletes the dynamic SSL configuration selection from the configuration.

Target object

None.

Required parameters

-dynSSLConfigSelectionName
Specifies the name that uniquely identifies the dynamic SSL configuration selection. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteDynamicSSLConfigSelection {-dynSSLConfigSelectionName sampleConfigSelection}
```

- Using Jython:

```
AdminTask.deleteDynamicSSLConfigSelection(-dynSSLConfigSelectionName sampleConfigSelection)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteDynamicSSLConfigSelection {-interactive}
```

- Using Jython:

```
AdminTask.deleteDynamicSSLConfigSelection('-interactive')
```

getDynamicSSLConfigSelection

The **getDynamicSSLConfigSelection** command obtains information about a particular dynamic SSL configuration selection.

Target object

None.

Required parameters

-dynSSLConfigSelectionName
Specifies the name that uniquely identifies the dynamic SSL configuration selection. (String, required)

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getDynamicSSLConfigSelection {-dynSSLConfigSelectionName sampleConfigSelection}
```

- Using Jython:

```
AdminTask.getDynamicSSLConfigSelection(-dynSSLConfigSelectionName sampleConfigSelection)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getDynamicSSLConfigSelection {-interactive}
```

- Using Jython:

```
AdminTask.getDynamicSSLConfigSelection('-interactive')
```

listDynamicSSLConfigSelections

The **listDynamicSSLConfigSelections** command lists the configuration objects name for a dynamic SSL configuration selection.

Target object

None.

Required parameters

None.

Optional parameters

-scopeName
Specifies the unique name that identifies the management scope. (String, optional)

-all
Specify the value of this parameter as `true` to list all dynamic SSL configuration selections. This parameter overrides the scopeName parameter. The default value is `false`. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listDynamicSSLConfigSelections
```

- Using Jython:

```
AdminTask.listDynamicSSLConfigSelections()
```

- Using Jacl:

```
$AdminTask listDynamic SSLConfigSelections
```

- Using Jython:

```
AdminTask.listDynamic SSLConfigSelections()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listDynamicSSLConfigSelections {-interactive}
```

- Using Jython:

```
AdminTask.listDynamicSSLConfigSelections('-interactive')
```

PersonalCertificateCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the PersonalCertificateCommands group can be used to create and manage personal or signer certificates.

The PersonalCertificateCommands command group for the AdminTask object includes the following commands:

createChainedCertificate

The createChainedCertificate command creates a new self-signed certificate and stores the certificate in a keystore.

Note: To use the IBMi5OSKeyStore key store, verify that the signer for each part of the chain exists in the keystore before creating the new certificate. You must import the signer into the IBMi5OSKeyStore keystore before creating the new certificate.

Target object

None.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
Specifies the name that uniquely identifies the certificate request in a keystore. (String, required)

-certificateSize
Specifies the size of the certificate. (Integer, required)

-certificateCommonName
Specifies the common name of the certificate. (String, required)

-certificateOrganization
Specifies the organization of the certificate. (String, optional)

Optional parameters

-rootCertificateAlias
Specifies a unique name to identify the root certificated to use for signing. The default root certificate alias is `root`. (String, optional)

-certificateVersion
Specifies the version of the certificate. (String, optional)

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-certificateOrganization
Specifies the organization of the certificate. (String, optional)

-certificateOrganizationalUnit
Specifies the organizational unit of the certificate. (String, optional)

-certificateLocality
Specifies the locality of the certificate. (String, optional)

-certificateState
Specifies the state of the certificate. (String, optional)

-certificateZip
Specifies the zip code of the certificate. (String, optional)

-certificateCountry
Specifies the country of the certificate. (String, optional)

-certificateValidDays
Specifies the amount of time in days for which the certificate is valid. (Integer, optional)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.createChainedCertificate('-keyStoreName myKeystore -certificateAlias
newCertificate -certificateSize 10 -certificateCommonName localhost
-certificateOrganization ibm')
```

- Using Jython list:

```
AdminTask.createChainedCertificate('-keyStoreName', 'myKeystore', '-certificateAlias',
'newCertificate', '-certificateSize', '10', '-certificateCommonName', 'localhost',
'-certificateOrganization', 'ibm')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createChainedCertificate('-interactive')
```

createSelfSignedCertificate

The createSelfSignedCertificate command creates a self-signed personal certificate in a keystore.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

-certificateVersion
The version of the certificate. (String, required)

-certificateSize
The size of the certificate. (Integer, required)

-certificateCommonName
The common name of the certificate. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

-certificateOrganization
The organization of the certificate. (String, optional)

-certificateOrganizationalUnit
The organizational unit of the certificate. (String, optional)

-certificateLocality
The locality of the certificate. (String, optional)

-certificateState
The state of the certificate. (String, optional)

-certificateZip
The zip code of the certificate. (String, optional)

-certificateCountry
The country of the certificate. (String, optional)

-certificateValidDays
The amount of time in days for which the certificate is valid. (Integer, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createSelfSignedCertificate {-keyStoreName testKeyStore -certificateAlias
default -certificateCommonName localhost -certificateOrganization ibm}
```

- Using Jython string:

```
AdminTask.createSelfSignedCertificate('[-keyStoreName testKeyStore -certificateAlias
default -certificateCommonName localhost -certificateOrganization ibm]')
```

- Using Jython list:

```
AdminTask.createSelfSignedCertificate(['-keyStoreName', 'testKeyStore', '-certificateAlias',
'default', '-certificateCommonName', 'localhost', '-certificateOrganization', 'ibm'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.createSelfSignedCertificate('-interactive')
```

deleteCertificate

The deleteCertificate command deletes a personal certificate from a keystore. The command saves a copy of the certificate in the delete keystore.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Interactive mode example usage:

- Using Jython:

```
AdminTask.deleteCertificate('-interactive')
```

exportCertificate

The exportCertificate command exports a personal certificate from one keystore to another.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-keyStorePassword
The password to the keystore. (String, required)

-keyFilePath
The full path to a keystore file that is located in a file system. The store from where a certificate will be imported or exported. (String, required)

-keyFilePassword
The password to the keystore file. (String, required)

-keyFileType
The type of the key file. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

-aliasInKeyStore
(String, optional)

Example output

The command does not return output.

Examples

Interactive mode example usage:

- Using Jython:

```
AdminTask.exportCertificate('-interactive')
```

exportCertToManagedKS

The exportCertToManagedKS command exports a personal certificate to a managed keystore in the configuration.

Target object

None.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore configuration object. (String, required)

-keyStorePassword
The password to the keystore. (String, required)

-toKeyStoreName
Specifies the unique name of the keystore to export the certificate to. (String, required)

-certificateAlias
Specifies the alias of the certificate of interest. (String, required)

Optional parameters

-keyStoreScope
Specifies the keystore of the certificate of interest. (String, optional)

-toKeyStoreScope
Specifies the scope of the keystore to export to. (String, optional)

-aliasInKeyStore
Specifies the alias that identifies the certificate in the keystore. (String, optional)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportCertificateToManagedKS('-keyStoreName myKS -keyStorePassword myKSpw
-toKeyStoreName myKS2 -certificateAlias testingKeyStore')
```

- Using Jython list:

```
AdminTask.exportCertificateToManagedKS(['-keyStoreName', 'myKS', '-keyStorePassword',
'myKSpw', '-toKeyStoreName', 'myKS2', '-certificateAlias', 'testingKeyStore'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportCertificateToManagedKS('-interactive')
```

extractCertificate

The extractCertificate command extracts the signer part of a personal certificate to a certificate file. The certificate in the file can later be added to a keystore to establish trust.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

-certificateFilePath
The full path of the request file that contains the certificate. (String, required)

-base64Encoded
Set the value of this parameter to `true` if the certificate is a Base64 encoded ASCII file type. Set the value of this parameter to `false` if the certificate is binary. (Boolean, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask extractCertificate {-keyStoreName testKeyStore -certificateFilePath
/temp/CertFile.arm -certificateAlias testCertificate}
```

- Using Jython string:

```
AdminTask.extractCertificate('[-keyStoreName testKeyStore -certificateFilePath
/temp/CertFile.arm -certificateAlias testCertificate]')
```

- Using Jython list:

```
AdminTask.extractCertificate(['-keyStoreName', 'testKeyStore', '-certificateFilePath',
'/temp/CertFile.arm', '-certificateAlias', 'testCertificate'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.extractCertificate('-interactive')
```

getCertificate

The getCertificate command obtains information about a particular personal certificate in a keystore. If the certificate of interest was created with the requestCACertificate command, the certificate can be in the COMPLETE or REVOKED state. Certificate requests can be in the PENDING state. Use the getCertificateRequest command to determine if a certificate request is in the PENDING state.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

Example output

The command returns information about the certificate request.

Examples

Interactive mode example usage:

- Using Jython:

```
AdminTask.getCertificate('-interactive')
```

getCertificateChain

The getCertificateChain command queries your configuration for information about each personal certificate in a certificate chain.

Target object

None.

Required parameters and return values

-keyStoreName
Specifies the name of the keystore object that stores the CA certificate. Use the listKeyStores command to display a list of available keystores. (String, required)

-certificateAlias
Specifies the unique alias of the certificate. (String, required)

Optional parameters

-keyStoreScope
Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope. (String, optional)

Example output

The command returns an array of attribute lists that contain configuration information for each certificate in a chain.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getCertificateChain {-certificateAlias newCertificate
-keyStoreName CellDefaultKeyStore}
```

- Using Jython string:

```
AdminTask.getCertificateChain('-certificateAlias newCertificate
-keyStoreName CellDefaultKeyStore')
```

- Using Jython list:

```
AdminTask.getCertificateChain(['-certificateAlias', 'newCertificate',
'-keyStoreName', 'CellDefaultKeyStore'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.getCertificateChain('-interactive')
```

importCertificate

The importCertificate command imports a personal certificate from a keystore.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-keyFilePath
The full path to a keystore file that is located in a file system. The store from where a certificate will be imported or exported. (String, required)

-keyFilePassword
The password to the keystore file. (String, required)

-keyFileType
The type of the key file. (String, required)

-certificateAliasFromKeyFile
The certificate alias in the key file from which the certificate is being imported. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Interactive mode example usage:

- Using Jython:

```
AdminTask.importCertificate('-interactive')
```

importCertFromManagedKS

The importCertFromManagedKS command imports a personal certificate from a managed keystore in the configuration.

Target object

None.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore configuration object. (String, required)

-fromKeyStoreName
Specifies the name that uniquely identifies the keystore from which the system imports the certificate. (String, required)

-fromKeyStorePassword
Specifies the password for the keystore from which the system imports the certificate. (String, required)

-certificateAliasFromKeyStore
Specifies the alias of the certificate in the keystore. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope of the keystore to import the certificate to. (String, optional)

-fromKeyStoreScope
Specifies the scope of the keystore to import the certificate from. (String, optional)

-certificateAlias
Specifies the alias of the certificate for the destination keystore. (String, optional)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.importCertFromManagedKS('-keyStoreName myKeystore -fromKeyStoreName
oldKeystore -fromKeyStorePassword my122password -certificateAliasFromKeyStore
myCertificate')
```

- Using Jython list:

```
AdminTask.importCertFromManagedKS('-keyStoreName', 'myKeystore', '-fromKeyStoreName',
'oldKeystore', '-fromKeyStorePassword', 'my122password', '-certificateAliasFromKeyStore',
'myCertificate')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importCertFromManagedKS('-interactive')
```

listKeySizes

The listKeySizes command is an administrative console helper task used to display the set of certificate key sizes that are allowed when creating a personal certificate.

By default, the set of valid key sizes include 512, 1024, 2048, 4096 and 8192. You can customize the list of comma-separated key sizes in the com.ibm.websphere.customKeySizeList security custom property. The com.ibm.websphere.customKeySizeList custom property can be set using the setAdminActiveSecuritySettings command.

Valid key sizes must be 512 or larger, but no larger then 16384, and must be multiples of 8. Any values in com.ibm.websphere.customKeySizeList that do not meet the size requirements are ignored. If the custom list does not contain anything in it when it is finished processing then the list of default sizes is returned.

Target object

None.

Required parameters

None.

Optional parameters

None.

Example output

Returns an arrayList of sizes. The sizes are strings:

512
1024
2048
4096
8192

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listKeySize
```

- Using Jython string:

```
AdminTask.listKeySizes()
```

listPersonalCertificates

The listPersonalCertificates command lists the personal certificates in a particular keystore.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. The value of this field is not a path to the keystore file. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. To obtain a list of the keystore scope values, see the listManagementScopes command, which is part of the ManagementScopeCommands command group. (String, optional)

Example output

The command returns a list of attributes for each personal certificate in a keystore.

Examples

Batch mode example usage:

- Using Jython string:

```
AdminTask.listPersonalCertificates('-keyStoreName myKS')
```

- Using Jython list:

```
AdminTask.listPersonalCertificates(['-keyStoreName', 'myKS'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.listPersonalCertificates('-interactive')
```

queryCACertificate

The queryCACertificate command queries your configuration to determine if the CA has completed the certificate. If the CA returns a personal certificate, then the system marks the certificate as `COMPLETE`. Otherwise, it remains marked as `PENDING`.

Target object

None.

Required parameters and return values

-keyStoreName
Specifies the name of the keystore object that stores the CA certificate. Use the listKeyStores command to display a list of available keystores. (String, required)

-certificateAlias
Specifies the unique alias of the certificate. (String, required)

Optional parameters

-keyStoreScope
Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope. (String, optional)

Example output

The command returns one of two values: `Certificate COMPLETE` or `certificate PENDING`. If the command returns the `Certificate COMPLETE` message, the certificate authority returned the requested certificate and the default personal certificate is replaced. If the command returns the `certificate PENDING` message, the certificate authority did not yet return a certificate.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask queryCACertificate {-certificateAlias newCertificate
-keyStoreName CellDefaultKeyStore}
```

- Using Jython string:

```
AdminTask.queryCACertificate('-certificateAlias newCertificate
-keyStoreName CellDefaultKeyStore')
```

- Using Jython list:

```
AdminTask.queryCACertificate(['-certificateAlias', 'newCertificate',
'-keyStoreName', 'CellDefaultKeyStore'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.queryCACertificate('-interactive')
```

receiveCertificate

The receiveCertificate command receives a signer certificate from a file to a personal certificate.

Target object

None.

Required parameters

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

-certificateFilePath
The full path of the file that contains the certificate. (String, required)

-base64Encoded
Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (Boolean, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask receiveCertificate {-keyStoreName testKeyStore
-certificateFilePath /temp/CertFile.arm}
```

- Using Jython string:

```
AdminTask.receiveCertificate('[-keyStoreName testKeyStore
-certificateFilePath /temp/CertFile.arm]')
```

- Using Jython list:

```
AdminTask.receiveCertificate(['-keyStoreName', 'testKeyStore',
'-certificateFilePath', '/temp/CertFile.arm'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.receiveCertificate('-interactive')
```

renewCertificate

The renewCertificate command renews a certificate with a new generated certificate.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name that identifies the keystore. (String, required)

-certificateAlias
Specifies the unique name that identifies the certificate. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope of the keystore. (String, optional)

-deleteOldSigners
Specifies whether to delete the old signers that are associated with the old certificate. Specify `false` to retain the old signers. (Boolean, optional)

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.renewCertificate('-keyStoreName myKS -certificateAlias
testCertificate')
```

- Using Jython list:

```
AdminTask.renewCertificate(['-keyStoreName', 'myKS', '-certificateAlias',
'testCertificate'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.renewCertificate('-interactive')
```

replaceCertificate

The replaceCertificate command replaces a personal certificate with another personal certificate. The command finds each reference to the old certificate alias in the configuration and replaces the alias with the new one. The command also replaces each signer certificate from the old personal certificate with the signer from the new personal certificate.

Target object

None.

Required parameters and return values

-keyStoreName
The name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
The name that uniquely identifies the certificate request in a keystore. (String, required)

-replacementCertificateAlias
The alias of the certificate that is used to replace a different certificate. (String, required)

Optional parameters

-keyStoreScope
The scope name of the keystore. (String, optional)

-deleteOldCert
Set the value of this parameter to `true` if you want to delete the old signer certificates during certificate replacement. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

-deleteOldSigners
Set the value of this parameter to `true` if you want to delete the old certificates during certificate replacement. Otherwise, set the value of this parameter to `false`. (Boolean, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask replaceCertificate {-keyStoreName testKeyStore -certificateAlias
default -replacementCertificateAlias replaceCert -deleteOldCert true
-deleteOldSigners true}
```

- Using Jython string:

```
AdminTask.replaceCertificate('[-keyStoreName testKeyStore -certificateAlias
default -replacementCertificateAlias replaceCert -deleteOldCert true
-deleteOldSigners true]')
```

- Using Jython list:

```
AdminTask.replaceCertificate(['-keyStoreName', 'testKeyStore', '-certificateAlias',
'default', '-replacementCertificateAlias', 'replaceCert', '-deleteOldCert',
'true', '-deleteOldSigners', 'true'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.replaceCertificate('-interactive')
```

requestCACertificate

The requestCACertificate command creates a certificate request and sends the request to a certificate authority (CA). If the certificate authority returns a personal certificate, then the returned certificate replaces the certificate request in the keystore. The command also works with a preexisting certificate request that was created with the createCertificateRequest command. When the CA returns a personal certificate, the system marks the certificate as `COMPLETE` and the command returns a message stating that the certificate is complete. If the CA does not return a personal certificate, then the system marks the certificate request as `PENDING` and the command returns a message stating that the certificate is `PENDING`.

Note: To use the IBMi5OSKeyStore key store, verify that the signer for each part of the chain exists in the keystore before creating the new certificate. You must import the signer into the IBMi5OSKeyStore keystore before creating the new certificate.

Target object

None.

Required parameters and return values

-certificateAlias
Specifies the alias of the certificate. You can specify a predefined certificate request. (String, required)

-keyStoreName
Specifies the name of the keystore object that stores the CA certificate. Use the listKeyStores command to display a list of available keystores. (String, required)

-caClientName
Specifies the name of the CA client that was used to create the CA certificate. (String, required)

-revocationPassword
Specifies the password to use to revoke the certificate at a later date. (String, required)

Optional parameters

-keyStoreScope
Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope. (String, optional)

-caClientScope
Specifies the management scope of the CA client. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope. (String, optional)

-certificateCommonName
Specifies the common name (CN) part of the full distinguished name (DN) of the certificate. This common name can represent a person, company, or machine. For websites, the common name is frequently the DNS host name where the server resides. (String, optional)

-certificateOrganization
Specifies the organization part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateOrganizationalUnity
Specifies the organization unit part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateLocality
Specifies the locality part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateState
Specifies the state part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateZip
Specifies the zip code part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateCountry
Specifies the country part of the full distinguished name (DN) of the certificate. (String, optional)

-certificateSize
Specifies the size of the certificate key. The valid values are 512, 1024, 2048, 4096, and 8192. The default value is 2048. (String, optional)

Example output

The command returns one of two values: `Certificate COMPLETE` or `certificate PENDING`.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask requestCACertificate {-certificateAlias newCertificate -keyStoreName
CellDefaultKeyStore -CAClientName myCAClient -revocationPassword revokeCApw}
```

- Using Jython string:

```
AdminTask.requestCACertificate('-certificateAlias newCertificate -keyStoreName
CellDefaultKeyStore -CAClientName myCAClient -revocationPassword revokeCApw')
```

- Using Jython list:

```
AdminTask.requestCACertificate(['-certificateAlias','newCertificate','-keyStoreName',
'CellDefaultKeyStore','-CAClientName','myCAClient','-revocationPassword',
'revokeCApw'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.requestCACertificate('-interactive')
```

revokeCACertificate

The revokeCACertificate command sends a request to the CA to revoke the CA personal certificate of interest.

Target object

None.

Required parameters and return values

-certificateAlias
Specifies the unique name that identifies the CA personal certificate object and the alias name of the certificate in the keystore. (String, required)

-keyStoreName
Specifies the name of the keystore where the CA personal certificate is stored. (String, required)

-revocationPassword
Specifies the password needed to revoke the certificate. This is the same password that was provided when the certificate was created. (String, required)

Optional parameters

-keyStoreScope
Specifies the management scope of the keystore. For a deployment manager profile, the default value is the cell scope. For an application server profile, the default value is the node scope. (String, optional)

-revocationReason
Specifies the reason for revoking the certificate of interest. The default value for this parameter is `unspecified`. (String, optional)

Example output

The command does not return output. Use the getCertificate command to view the current status of the certificate, as the following example displays:

```
AdminTask.getCertificate('-certificateAlias myCertificate -keyStoreName CellDefaultKeyStore')
```

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask revokeCACertificate {-keyStoreName CellDefaultKeyStore -certificateAlias
myCertificate -revocationPassword pw4revoke}
```

- Using Jython string:

```
AdminTask.revokeCACertificate('[-keyStoreName CellDefaultKeyStore -certificateAlias
myCertificate -revocationPassword pw4revoke]')
```

- Using Jython list:

```
AdminTask.revokeCACertificate(['-keyStoreName', 'CellDefaultKeyStore', '-certificateAlias',
'myCertificate', '-revocationPassword', 'pw4revoke'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.revokeCACertificate('-interactive')
```

WSCertExpMonitorCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the WSCertExpMonitorCommands group can be used to start or update the certificate expiration monitor.

The WSCertExpMonitorCommands command group for the AdminTask object includes the following commands:

- "createWSCertExpMonitor"
- "deleteWSCertExpMonitor" on page 260
- "getWSCertExpMonitor" on page 261
- "listWSCertExpMonitor" on page 262
- "modifyWSCertExpMonitor" on page 262
- "startCertificateExpMonitor" on page 263

createWSCertExpMonitor

The **createWSCertExpMonitor** command creates the certificate expiration monitor settings in the configuration.

Target object

None.

Required parameters and return values

-name
The name that uniquely identifies the certificate expiration monitor. (String, required)

-autoReplace
Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required)

-deleteOld
Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required)

-daysBeforeNotification
The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required)

-wsScheduleName
The name of the scheduler to use for certificate expiration. (String, required)

-wsNotificationName
The name of the notifier to use for certificate expiration. (String, required)

-isEnabled
Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional)

- Returns: The configuration object name of the certificate expiration monitor object that you created.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createWSCertExpMonitor {-name testCertMon –autoReplace true –deleteOld true
–daysBeforeNotification 30 –wsScheduleName testSchedule –wsNotificationName testNotifier
–isEnabled false}
```

- Using Jython string:

```
AdminTask.createWSCertExpMonitor ('[-name testCertMon –autoReplace true –deleteOld true
–daysBeforeNotification 30 –wsScheduleName testSchedule –wsNotificationName testNotifier
–isEnabled false]')
```

- Using Jython list:

```
AdminTask.createWSCertExpMonitor (['-name', 'testCertMon', '–autoReplace', 'true', '–deleteOld',
'true', '–daysBeforeNotification', '30', '–wsScheduleName', 'testSchedule', '–wsNotificationName',
'testNotifier', '–isEnabled', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createWSCertExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.createWSCertExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createWSCertExpMonitor (['-interactive'])
```

deleteWSCertExpMonitor

The **deleteWSCertExpMonitor** command deletes the settings of a scheduler from the configuration.

Target object

None.

Required parameters and return values

-name
The name that uniquely identifies the certificate expiration monitor. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteWSCertExpMonitor {-name testCertMon}
```

- Using Jython string:

```
AdminTask.deleteWSCertExpMonitor ('[-name testCertMon]')
```

- Using Jython list:

```
AdminTask.deleteWSCertExpMonitor (['-name', 'testCertMon'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteWSCertExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.deleteWSCertExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteWSCertExpMonitor (['-interactive'])
```

getWSCertExpMonitor

The **getWSCertExpMonitor** command displays the settings of a particular scheduler.

Target object

None.

Required parameters and return values

-name
The name that uniquely identifies the certificate expiration monitor. (String, required)

- Returns: The scheduler in the configuration.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getWSCertExpMonitor {-name testCertMon}
```

- Using Jython string:

```
AdminTask getWSCertExpMonitor ('[-name testCertMon]')
```

- Using Jython list:

```
AdminTask getWSCertExpMonitor (['-name', 'testCertMon'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getWSCertExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.getWSCertExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getWSCertExpMonitor (['-interactive'])
```

listWSCertExpMonitor

The **listWSCertExpMonitor** command lists the scheduler in the configuration.

Target object

None.

Required parameters and return values

-displayObjectNames
If you set the value of this parameter to true, the command returns the certificate expiration monitor configuration object. If you set the value of this parameter to false, the command returns the name of the certificate expiration monitor. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listWSCertExpMonitor {-displayObjectName false}
```

- Using Jython string:

```
AdminTask.listWSCertExpMonitor ('[-displayObjectName false]')
```

- Using Jython list:

```
AdminTask.listWSCertExpMonitor (['-displayObjectName', 'false'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listWSCertExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.listWSCertExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listWSCertExpMonitor (['-interactive'])
```

modifyWSCertExpMonitor

The **modifyWSCertExpMonitor** command changes the setting of an existing scheduler.

Target object

None.

Required parameters and return values

-name
The name that uniquely identifies the certificate expiration monitor. (String, required)

-autoReplace
Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required)

-deleteOld
Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required)

-daysBeforeNotification
The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required)

-wsScheduleName
The name of the scheduler to use for certificate expiration. (String, required)

-wsNotificationName
The name of the notifier to use for certificate expiration. (String, required)

-isEnabled
Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional)

- Returns: None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifyWSCertExpMonitor {-name testCertMon –autoReplace false –deleteOld false
–daysBeforeNotification 20 –isEnabled true}
```

- Using Jython string:

```
AdminTask.modifyWSCertExpMonitor ('[-name testCertMon –autoReplace false –deleteOld false
–daysBeforeNotification 20 –isEnabled true]')
```

- Using Jython list:

```
AdminTask.modifyWSCertExpMonitor (['-name', 'testCertMon', '–autoReplace', 'false', '–deleteOld',
'false', '–daysBeforeNotification', '20', '–isEnabled', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyWSCertExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.modifyWSCertExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.modifyWSCertExpMonitor (['-interactive'])
```

startCertificateExpMonitor

The **startCertificateExpMonitor** command performs certificate monitoring. This command visits all key stores and checks to see if they are within certificate expiration range.

Target object

None.

Required parameters and return values

- Parameters: None
- Returns: None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask startCertificateExpMonitor
```

- Using Jython:

```
AdminTask.startCertificateExpMonitor()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask startCertificateExpMonitor {-interactive}
```

- Using Jython string:

```
AdminTask.startCertificateExpMonitor ('[-interactive]')
```

- Using Jython list:

```
AdminTask.startCertificateExpMonitor (['-interactive'])
```

SignerCertificateCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the SignerCertificateCommands group can be used to create and modify signer certificates in relation to the key store file and to query for signer information on ports of remote hosts.

The SignerCertificateCommands command group for the AdminTask object includes the following commands:

addSignerCertificate

The **addSignerCertificate** command add a signer certificate from a certificate file to a keystore.

Target object

None.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
Specifies the name that uniquely identifies the certificate request in a keystore. (String, required)

-certificateFilePath
Specifies the full path of the request file that contains the certificate. (String, required)

-base64Encoded
Specifies that the certificate is a Base64 encoded ASCII data file type if the value is set to `true`. Set the value of this parameter to `false` if the certificate is a binary DER data file type. (Boolean, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addSignerCertificate {-keyStoreName testKeyStore -certificateAlias
default -certificateFilePath <file path> -base64Encoded true}
```

- Using Jython string:

```
AdminTask.addSignerCertificate('[-keyStoreName testKeyStore -certificateAlias
default -certificateFilePath <file path> -base64Encoded true]')
```

- Using Jython list:

```
AdminTask.addSignerCertificate(['-keyStoreName', 'testKeyStore', '-certificateAlias',
'default', '-certificateFilePath', '<file path>', '-base64Encoded', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addSignerCertificate {-interactive}
```

- Using Jython string:

```
AdminTask.addSignerCertificate ('[-interactive]')
```

deleteSignerCertificate

The **deleteSignerCertificate** command delete a signer certificate from a certificate file from a keystore.

Target object

None.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore configuration object. (String, required)

-certificateAlias
Specifies the name that uniquely identifies the certificate request in a keystore. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

Example output

The command does not return output.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteSignerCertificate {-keyStoreName testKeyStore -certificateAlias
default}
```

- Using Jython string:

```
AdminTask.deleteSignerCertificate('[-keyStoreName testKeyStore -certificateAlias
default]')
```

- Using Jython list:

```
AdminTask.deleteSignerCertificate(['-keyStoreName', 'testKeyStore', '-certificateAlias',
'default'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteSignerCertificate {-interactive}
```

- Using Jython string:

```
AdminTask.deleteSignerCertificate ('[-interactive]')
```

extractSignerCertificate

The **extractSignerCertificate** command extracts a signer certificate from a key store to a file.

Target object

None

Parameters and return values

-keyStoreName
The name of the key store where the signer certificate is located. (String, required)

-keyStoreScope
The management scope of the key store. (String, optional)

-certificateAlias
The alias name of the signer certificate in the key store. (String, required)

-certificateFilePath
The full path name of the file that contains the signer certificate. (String, required)

-base64Encoded
Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (String, required)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask extractSigner Certificate {-interactive}
```

- Using Jython string:

```
AdminTask.extractSignerC ertificate ('[-interactive]')
```

- Using Jython list:

```
AdminTask.extractSignerCe rtificate (['-interactive'])
```

getSignerCertificate

The **getSignerCertificate** command obtains information about a signer certificate from a key store.

Target object

None

Parameters and return values

-keyStoreName
The name of the key store where the signer certificate is located. (String, required)

-keyStoreScope
The management scope of the key store. (String, optional)

-certificateAlias
The alias name of the signer certificate in the key store. (String, required)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getSignerCert ificate {-interactive}
```

- Using Jython string:

```
AdminTask.getSignerCert ificate ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getSignerCert ificate (['-interactive'])
```

listSignerCertificates

The **listSignerCertificates** command lists all signer certificates in a particular key store.

Target object

None

Parameters and return values

-keyStoreName
The name of the key store where the signer certificate is located. (String, required)

-keyStoreScope
The management scope of the key store. (String, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listSignerCer tificates {-interactive}
```

- Using Jython string:

```
AdminTask.listSignerCer tificates ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listSignerCer tificates (['-interactive'])
```

retrieveSignerFromPort

The **retrieveSignerFromPort** command retrieves a signer from a remote host and stores the signer in a key store.

Target object

None

Parameters and return values

-host
The host name of the system from where the signer certificate will be retrieved. (String, required)

-port
The port of the remote system from where the signer certificate will be retrieved. (Integer, required)

-certificateAlias
Specifies a unique name to identify a certificate. (String, required)

-keyStoreName
The name of the key store where the signer certificate is located. (String, required)

-keyStoreScope
The management scope of the key store. (String, optional)

-sslConfigName
The name of the SSL configuration object. (String, optional)

-sslConfigScopeName
The management scope where the SSL configuration object is located. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask retrieveSigner FromPort {-host serverHost -port 443 -keyStoreName testKeyStore
-certificate Alias serverHostSigner}
```

- Using Jython string:

```
AdminTask.retrieveSigner FromPort ('[-host server Host -port 443 -keyStore Name testKeyStore
-certi ficateAlias serverHost Signer]')
```

- Using Jython list:

```
AdminTask.retrieveSigner FromPort (['-host', 'serverHost', '-port', '443', '-keyStoreName',
'testKeyStore', '-certi ficateAlias', 'serverHost Signer'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask retrieveSigner FromPort {-interactive}
```

- Using Jython string:

```
AdminTask.retrieveSigner FromPort ('[-interactive]')
```

- Using Jython list:

```
AdminTask.retrieveSigner FromPort (['-interactive'])
```

retrieveSignerInfoFromPort

The **retrieveSigner InfoFromPort** command retrieves signer information from a port on a remote host.

Target object

None

Parameters and return values

-host
The host name of the system from where the signer certificate will be retrieved. (String, required)

-port
The port of the remote system from where the signer certificate will be retrieved. (Integer, required)

-sslConfigName
The name of the SSL configuration object. (String, optional)

-sslConfigScopeName
The management scope where the SSL configuration object is located. (String, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask retrieveSigner InfoFromPort {-interactive}
```

- Using Jython string:

```
AdminTask.retrieveSigner InfoFromPort ('[-interactive]')
```

- Using Jython list:

```
AdminTask.retrieveSigner InfoFromPort (['-interactive'])
```

CertificateRequestCommands command group of the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the CertificateRequestCommands group can be used to create and manage certificate requests.

The CertificateRequestCommands command group for the AdminTask object includes the following commands:

createCertificateRequest

The **createCertificateRequest** command creates a certificate request that is associated with a particular key store.

Target object

None.

Parameters and return values

-keyStoreName
The name that uniquely identifies the key store configuration object. (String, required)

-keyStoreScope
The scope name of the key store. (String, optional)

-certificateAlias
The name that uniquely identifies the certificate request in a key store. (String, required)

-certificateVersion
The certificate version. (String, required)

-certificateSize
(Integer, required)

-certificateCommonName
(String, required)

-certificateOrganization
(String, optional)

-certificateOrganizationalUnit
(String, optional)

-certificateLocality
(String, optional)

-certificateState
The state code for the certificate. (String, optional)

-certificateZip
The zip code for the certificate. (String, optional)

-certificateCountry
The country for the certificate. (String, optional)

-certificateValidDays
The amount of time in days for which the certificate is valid. (Integer, optional)

-certificateRequestFilePath
The file location of the certificate request that can be sent to a certificate authority. (String, required)

- Returns: The configuration object name of the key store object that you created.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createCertificateRequest {-keyStoreName testKeyStore
-certificateAlias certReq -certificateSize 1024 -certificate
CommonName localhost -certificate Organization testing -certificate
RequestFilePath c:/temp/testCertReq.arm}
```

- Using Jython string:

```
AdminTask.createCertificateRequest ('[-keyStoreName testKeyStore
-certificateAlias certReq -certificateSize 1024 -certificate
CommonName localhost -certificate Organization testing -certificate
RequestFilePath c:/temp/testCertReq.arm]')
```

- Using Jython list:

```
AdminTask.createCertificateRequest (['-keyStoreName', 'testKeyStore',
'-certificateAlias', 'certReq', '-certificateSize', '1024',
'-certificateCommonName', 'localhost','-certificateOrganization',
'testing', '-certificateRequestFilePath','c:/temp/testCertReq.arm'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createCertificateRequest {-interactive}
```

- Using Jython string:

```
AdminTask.createCertificateRequest ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createCertificateRequest (['-interactive'])
```

deleteCertificateRequest

The **deleteCertificateRequest** command deletes a certificate request from a key store.

Target object

None.

Parameters and return values

-keyStoreName
The name that uniquely identifies the key store configuration object. (String, required)

-keyStoreScope
The scope name of the key store. (String, optional)

-certificateAlias
The name that uniquely identifies the certificate request in a key store. (String, required)

- Returns: None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteCertificateRequest {-keyStoreName testKeyStore
-certificateAlias certReq}
```

- Using Jython string:

```
AdminTask.deleteCertificateRequest ('[-keyStoreName testKeyStore
-certificateAlias certReq]')
```

- Using Jython list:

```
AdminTask.deleteCertificateRequest (['-keyStoreName', 'testKeyStore',
'-certificateAlias', 'certReq'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteCertificateRequest {-interactive}
```

- Using Jython string:

```
AdminTask.deleteCertificateRequest ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteCertificateRequest (['-interactive'])
```

extractCertificateRequest

The **extractCertificateRequest**command extracts a certificate request to a file.

Target object

None.

Parameters and return values

-keyStoreName
The name that uniquely identifies the key store configuration object. (String, required)

-keyStoreScope
The scope name of the key store. (String, optional)

-certificateAlias
The name that uniquely identifies the certificate request in a key store. (String, required)

-certificateRequestFilePath
The file location of the certificate request that can be sent to a certificate authority. (String, required)

- Returns: A certificate request file is created that contains the extracted certificate.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask extractCertificateRequest {-keyStoreName testKeyStore
-certificateAlias certReq -certificateRequestFilePath c:/temp/testCertReq.arm}
```

- Using Jython string:

```
AdminTask.extractCertificateRequest ('[-keyStoreName testKeyStore
-certificateAlias certReq -certificateRequestFilePath c:/temp/testCertReq.arm]')
```

- Using Jython list:

```
AdminTask.extractCertificateRequest (['-keyStoreName', 'testKeyStore',
'-certificateAlias', 'certReq', '-certificateRequestFilePath','c:/temp/testCertReq.arm'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask extractCertificateRequest {-interactive}
```

- Using Jython string:

```
AdminTask.extractCertificateRequest ('[-interactive]')
```

- Using Jython list:

```
AdminTask.extractCertificateRequest (['-interactive'])
```

getCertificateRequest

The **getCertificateRequest** command obtains information about a particular certificate request in a key store.

Target object

None.

Parameters and return values

-keyStoreName
The name that uniquely identifies the key store configuration object. (String, required)

-keyStoreScope
The scope name of the key store. (String, optional)

-certificateAlias
The name that uniquely identifies the certificate request in a key store. (String, required)

- Returns: Information about the certificate request.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getCertificateRequest {-keyStoreName testKeyStore
-certificateAlias certReq}
```

- Using Jython string:

```
AdminTask.getCertificateRequest ('[-keyStoreName testKeyStore
-certificateAlias certReq]')
```

- Using Jython list:

```
AdminTask.getCertificateRequest (['-keyStoreName', 'testKeyStore',
'-certificateAlias', 'certReq'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getCertificateRequest {-interactive}
```

- Using Jython string:

```
AdminTask.getCertificateRequest ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getCertificateRequest (['-interactive'])
```

listCertificateRequests

The **listCertificateRequests** command lists all the certificate requests associated with a particular key store.

Target object

None.

Parameters and return values

-keyStoreName
The name that uniquely identifies the key store configuration object. (String, required)

-keyStoreScope
The scope name of the key store. (String, optional)

- Returns: An attribute list for each certificate request in a key store.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listCertificateRequest {-keyStoreName testKeyStore}
```

- Using Jython string:

```
AdminTask.listCertificateRequest ('[-keyStoreName testKeyStore]')
```

- Using Jython list:

```
AdminTask.listCertificateRequest (['-keyStoreName', 'testKeyStore'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listCertificateRequests {-interactive}
```

- Using Jython string:

```
AdminTask.listCertificateRequests ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listCertificateRequests (['-interactive'])
```

Enabling authentication in the file transfer service using scripting

The file transfer service provides role-based authentication. You can enable authentication in the file transfer service using scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

There are two versions of the file transfer Web application.; a secured version and an unsecured version. The secured version of this file, which is the version that authenticates its caller, is installed by default. The secured version is located in the *app_server_root*/systemApps/filetransferSecured.ear directory. The unsecured version, which is the version that does not authenticate its caller, is located in the *app_server_root*/systemApps/filetransfer.ear directory

In WebSphere Application Server a mixed cell environments, file transfer is a system application. You can activate authentication in the file transfer service by redeploying the file transfer Web application at the deployment manager level.

Procedure

- A wsadmin Jacl script is provided to help you redeploy the file transfer Web application. The script is called redeployFileTransfer.jacl and is located in the *app_server_root*/bin directory.

  Note: If you use more than one profile in your environment, use the **profile** parameter with this command to specify the profile from which to run the redeployFileTransfer.jacl wsadmin Jacl script. If you do not specify a profile path, the script uses the default profile. For example, you might specify *profile_root*/bin/redeployFileTransfer.jacl where *profile_root* is the path to the profile.

  You can deploy the secured file transfer service by running the script.

The syntax for running the script from the bin directory is the following:

–

```
wsadmin -conntype NONE -lang jacl -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationXxx
cell_name node_name server_name"
```

- *Xxx* is **On** or **Off**.

  To determine whether to turn file transfer authentication on or off, open the `systemApps.xml` file within the *profile_root*`/config/cells/`*cell_name*`/nodes/`*node_name*`/` directory. If you see a reference to the `filetransfer.ear` file within the `systemApps.xml` file, specify `fileTransferAuthenticationOn`. If you see a reference to the `filetransfersecured.ear` file within the `systemApps.xml` file, specify `fileTransferAuthenticationOff`.
- *cell_name* is the name of your cell.
- *node_name* is the name of your node.
- *server_name* is the name of your server.

Use wsadmin.

– For example, when running the script to enable use of the filetransferSecured.ear file, the syntax is similar to the following example:

```
wsadmin -conntype NONE -lang jacl -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOn managedCell managedCellManager dmgr"
```

or

```
wsadmin -conntype NONE -lang jacl -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOn baseCell base server1"
```

In the previous examples, the following values apply:

– *managedCell* and *baseCell* are the names of the cells.

– *managedCellManager* and *base* are the names of the nodes.

– *dmgr* and *server1* are the names of the servers.

• If you want to go return to running the file transfer service without authentication, you can run the script as shown in the following example:

```
wsadmin -conntype NONE -lang jacl -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOff baseNodeCell baseNode server1"
```

or

```
wsadmin -conntype NONE -lang jacl -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOff managedCell managedCellManager dmgr"
```

In the previous examples, the following values apply:

– *baseNodeCell* and *managedCell* are the names of the cells.

– *baseNode* and *managedCellManager* are the names of the nodes.

– *server1* and *dmgr* are the names of the servers.

What to do next

You must restart the server for the change to take affect.

Propagating security policy of installed applications to a JACC provider using wsadmin scripting

It is possible that you have applications installed prior to enabling the Java Authorization Contract for Containers (JACC)-based authorization. You can start with default authorization and then move to an external provider-based authorization using JACC later.

Before you begin

Note: Use the wsadmin tool to propagate information to the JACC provider independent of the application installation process, avoiding the need to reinstall applications. Also, during application installation

or modification you might have had problems propagating the security policy information to the JACC provider. For example, network problems might occur, the JACC provider might not be available, and so on. For these cases, the security policy of the previously installed applications does not exist in the JACC provider to make the access decisions. One choice is to reinstall the applications involved. However, you can avoid reinstalling by using the wsadmin scripting tool. Use this tool to propagate information to the JACC provider independent of the application installation process. The tool eliminates the need for reinstalling the applications.

The tool uses the SecurityAdmin MBean to propagate the policy information in the deployment descriptor of any installed application to the JACC provider. You can invoke this tool using wsadmin at the base application server for base and deployment manager level for WebSphere Application Server, Network Deployment. Note that the SecurityAdmin MBean is available only when the server is running.

Use `propagatePolicyToJACCProvider{-appNames appNames}` to propagate the policy information in the deployment descriptor or annotations of the enterprise archive (EAR) files to the JACC provider. If the RoleConfigurationFactory and the RoleConfiguration interfaces are implemented by the JACC provider, the authorization table information in the binding file of the EAR files is also propagated to the provider. See the *Securing applications and their environment* PDF for more information about these interfaces.

The `appNames String` contains the list of application names, delimited by a colon (:), whose policy information must be stored in the provider. If appNames is not present, the policy information of all the deployed applications is propagated to the provider.

Also, be aware of the following items:

- Before migrating applications to the Tivoli Access Manager JACC provider, create or import the users and groups that are in the applications to Tivoli Access Manager.
- Depending on the application or the number of applications that are propagated, you might have to increase the request time-out period either in the `soap.client.props` file in the directory *profile_root*`/properties` (if using SOAP) or in the `sas.client.props` file (if using RMI) for the command to complete. You can set the request time-out value to 0 to avoid the timeout problem, and change it back to the original value after the command is run.

Procedure

1. Configure your JACC provider in WebSphere Application Server.

   See the Authorizing access to J2EE resources using Tivoli Access Manager article for more information.

   See the *Securing applications and their environment* PDF for more information.
2. Restart the server.
3. Enter the following commands:

```
wsadmin>$AdminTask propagatePolicyToJACCProvider {-appNames appNames}
```

JACCUtilityCommands command group for the AdminTask object

Use this topic as a reference for the commands for the JACCUtilityCommands group for the AdminTask object. Use these commands to determine whether Java Authorization Contract for Containers (JACC) is enabled and whether the runtime uses a single security domain. You can also use these commands to propagate the security policies for application to the JACC provider.

The following commands are available for the JACCUtilityCommands group of the AdminTask object.

- “isJACCEnabled” on page 276
- “isSingleSecurityDomain” on page 276
- “propagatePolicyToJACCProvider ” on page 277

isJACCEnabled

The isJACCEnabled command displays whether JACC is enabled or disabled in the global security domain when the server was started. The command does not indicate dynamic changes. Instead, it displays the JACC status at server startup.

Target object

None.

Required parameters

None.

Return value

The command returns `true` if JACC is enabled. The command returns `false` if JACC is disabled.

Batch mode example usage

Using Jython string:

```
AdminTask.isJACCEnabled()
```

Interactive mode example usage

Using Jython:

```
AdminTask.isJACCEnabled('-interactive')
```

isSingleSecurityDomain

The isSingleSecurityDomain command displays whether the environment is configured to use a single security domain when the server was started. The command does not indicate dynamic changes. Instead, it displays the security domain status at server startup.

Target object

None.

Required parameters

None.

Return value

The command returns `true` if the environment uses a single security domain. The command returns the `false` string if the environment uses multiple security domains.

Batch mode example usage

Using Jython:

```
AdminTask.isSingleSecurityDomain()
```

Interactive mode example usage

Using Jython:

```
AdminTask.isSingleSecurityDomain('-interactive')
```

propagatePolicyToJACCProvider

The propagatePolicyToJACCProvider command propagates the security policies of the applications of interest to the JACC provider. This command is supported in a single security domain environment only.

Target object

None.

Required parameters

None.

Optional parameters

-appNames
Specifies a list of application names delimited with a colon character (:). (String, optional)

The command uses all applications if you do not specify a value for this parameter, as the following syntax demonstrates: `AdminTask.propagatePolicyToJACCProvider()`

Return value

The command does not return output.

Batch mode example usage

Using Jython string:

```
AdminTask.propagatePolicyToJACCProvider ('–appNames "app1:app2:app3"')
```

Using Jython list:

```
AdminTask.propagatePolicyToJACCProvider ('–appNames', '"app1:app2:app3"')
```

Interactive mode example usage

Using Jython:

```
AdminTask.propagatePolicyToJACCProvider ('-interactive')
```

Configuring custom adapters for federated repositories using wsadmin

You can use the Jython or Jacl scripting language with the wsadmin tool to define custom adapters in the federated repositories configuration file.

Before you begin

Shut down the WebSphere Application Server and the wsadmin command window.

About this task

The federated repositories configuration file, `wimconfig.xml`, is shipped with WebSphere Application Server 6.1.x and is located in the *app_server_root*/`profiles`/*profile_name*/`config`/`cells`/*cell_name*/`wim`/`config` directory.

Note: For additional information about the commands to use for this topic, see the IdMgrRepositoryConfig command group for the AdminTask object topic.

Use the following steps to add a custom adapter to any federated repositories configuration file and to any realm defined within the configuration file.

Procedure

1. Open the wimconfig.xml file with a text editor.
2. Add a new config:repositories element to the file. This element should be placed before the config:realmConfiguration element.

   The following example configures a custom repository to use the **com.ibm.ws.wim.adapter.sample.SampleFileAdapter** class and sets the **SampleFileRepository** repository as the identifier:

```
<config:repositories adapterClassName="com.ibm.ws.wim.adapter.sample.SampleFileAdapter"
id="SampleFileRepository"/>
```

3. Save the wimconfig.xml file and close the text editor.
4. Copy the vmmsampleadapter.jar file that is provided to app_server_root/lib.
5. Enter the following command to start the wsadmin tool:

```
wsadmin –conntype none
```

6. Disable paging in the common repository configuration. Set the supportPaging parameter for the updateIdMgrRepository command to false to disable paging.

   Note: You must perform this step because the sample adapter does not support paging.

   The following examples use the **SampleFileRepository** repository as the identifier for the custom repository.

   Using Jython:

```
AdminTask.updateIdMgrRepository('-id SampleFileRepository -supportPaging false')
```

   Using Jacl:

```
$AdminTask updateIdMgrRepository {-id SampleFileRepository -supportPaging
false}
```

   Note: A warning will appear until the configuration of the sample repository is complete.

7. Add the necessary custom properties for the adapter. Use the setIdMgrCustomProperty command repeatedly to add multiple properties. Use this command once per property to add multiple properties to your configuration. You must use both the **name** and **value** parameters to add the custom property for the specified repository. For example, to add a custom property of **fileName**, enter the following command.

   Using Jython:

```
AdminTask.setIdMgrCustomProperty('-id SampleFileRepository -name fileName
-value "c:\sampleFileRegistry.xml"')
```

   Using Jacl:

```
$AdminTask setIdMgrCustomProperty {-id SampleFileRepository -name fileName
-value "c:\sampleFileRegistry.xml"}
```

8. Add a base entry to the adapter configuration. Use the addIdMgrRepositoryBaseEntry command to specify the name of the base entry for the specified repository. For example:

   Using Jython:

```
AdminTask.addIdMgrRepositoryBaseEntry('-id SampleFileRepository -name
o=sampleFileRepository')
```

   Using Jacl:

```
$AdminTask addIdMgrRepositoryBaseEntry {-id SampleFileRepository -name
o=sampleFileRepository}
```

9. Use the addIdMgrRealmBaseEntry command to add the base entry to the realm, which will link the realm with the repository:

   Using Jython:

```
AdminTask.addIdMgrRealmBaseEntry('-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository')
```

   Using Jacl:

```
$AdminTask addIdMgrRealmBaseEntry {-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository}
```

10. Save your configuration changes. Enter the following commands to save the new configuration and close the wsadmin tool.

Using Jython:

```
AdminConfig.save()
exit
```

Using Jacl:

```
$AdminConfig save
exit
```

The following example displays the complete text of the newly-revised `wimconfig.xml` file:

```
<!--
  Begin Copyright

  Licensed Materials - Property of IBM

  virtual member manager

  (C) Copyright IBM Corp. 2005 All Rights Reserved.

  US Government Users Restricted Rights - Use, duplication or
  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

  End Copyright
-->
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:config="http://www.ibm.com/websphere/wim
/config" xmlns:sdo="commonj.sdo">
  <config:configurationProvider maxPagingResults="500" maxSearchResults="4500"
maxTotalPagingResults="1000"
pagedCacheTimeOut="900" pagingEntityObject="true" searchTimeOut="600000">
    <config:dynamicModel xsdFileName="wimdatagraph.xsd"/>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="Group">
      <config:rdnProperties>cn</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="OrgContainer">
      <config:rdnProperties>o</config:rdnProperties>
      <config:rdnProperties>ou</config:rdnProperties>
      <config:rdnProperties>dc</config:rdnProperties>
      <config:rdnProperties>cn</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="PersonAccount">
      <config:rdnProperties>uid</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:repositories xsi:type="config:FileRepositoryType" adapterClassName="com.ibm.
    ws.wim.adapter.file.was.FileAdapter"
id="InternalFileRepository" supportPaging="false" supportSorting="false" messageDigestAlgorithm="SHA-1">
      <config:baseEntries name="o=defaultWIMFileBasedRealm"/>
    </config:repositories>
    <config:repositories adapterClassName="com.ibm.ws.wim.adapter.sample.SampleFileAdapter"
    id="SampleFileRepository">
      <config:CustomProperties name="fileName" value="c:\sampleFileRegistry.xml"/>
      <config:baseEntries name="o=sampleFileRepository"/>
    </config:repositories>
    <config:realmConfiguration defaultRealm="defaultWIMFileBasedRealm">
      <config:realms delimiter="@" name="defaultWIMFileBasedRealm" securityUse="active">
        <config:participatingBaseEntries name="o=defaultWIMFileBasedRealm"/>
        <config:participatingBaseEntries name="o=sampleFileRepository"/>
        <config:uniqueUserIdMapping propertyForInput="uniqueName" propertyForOutput="uniqueName"/>
        <config:userSecurityNameMapping propertyForInput="principalName" propertyForOutput="principalName"/>
        <config:userDisplayNameMapping propertyForInput="principalName" propertyForOutput="principalName"/>
        <config:uniqueGroupIdMapping propertyForInput="uniqueName" propertyForOutput="uniqueName"/>
        <config:groupSecurityNameMapping propertyForInput="cn" propertyForOutput="cn"/>
        <config:groupDisplayNameMapping propertyForInput="cn" propertyForOutput="cn"/>
      </config:realms>
    </config:realmConfiguration>
</config:configurationProvider></sdo:datagraph>
```

11. Restart the application server.

Disabling embedded Tivoli Access Manager client using wsadmin

Follow these steps to unconfigure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager.

About this task

In a WebSphere Application Server, Network Deployment architecture, ensure that all the managed servers, including node agents, are started. Perform the following process once on the deployment management server. Details of the unconfiguration are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers require their own reboot for the configuration changes to take effect.

Note: It is also possible to unconfigure using the administrative console. For details on unconfiguring the embedded Tivoli Access Manager client using the WebSphere Application Server administrative console, refer to the Disabling embedded Tivoli Access Manager client using the administrative console article.

Procedure

1. Restart the deployment manager process.
2. Start the wsadmin command-line utility. The wsadmin command is found in the*install_dir*/bin directory
3. From the **wsadmin** prompt, enter the following command:

   ```
   WSADMIN>$AdminTask unconfigureTAM -interactive
   ```

   The following table lists the information that you are asked to provide for the unconfigureTAM command. The table also lists the properties that apply to the configureTAM and reconfigureTAM commands.

Table 39. Commands for configuring, reconfiguring, and unconfiguring Tivoli Access Manager. The following table lists the information that you are asked to provide for the configureTAM command. The table also lists the properties that apply to the unconfigureTAM and reconfigureTAM commands.

Property	Default	Relevant command	Description
Websphere Application Server node name	*	• configureTAM • reconfigureTAM • unconfigureTAM	Specify a single node or enter an asterisk (*) to run the configuration task on all of the application server instances including the deployment manager, node agents, and servers.
Tivoli Access Manager Policy Server	Default port: 7135	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration.
Tivoli Access Manager Authorization Server	Default port: 7136	• configureTAM • reconfigureTAM	Enter the name, port, and priority of each configured Tivoli Access Manager authorization server. Use the format *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration. You can specify more than one authorization server by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: auth_server1**:**7136**:**1,auth_server2**:**7137**:**2. A priority of 1 is still required when you use a single authorization server.
Websphere Application Server administrator's distinguished name		• configureTAM • reconfigureTAM	Enter the full distinguished name of the security primary administrator ID for WebSphere Application Server as created in the "Creating the security administrative user" topic in the *Securing applications and their environment* PDF. For example: cn=wasadmin,o=*organization*,c=*country*
Tivoli Access Manager user registry distinguished name suffix		• configureTAM • reconfigureTAM	Enter the suffix that you have set up in the user registry to contain the user and groups for Tivoli Access Manager. For example: o=*organization*,c=*country*
Tivoli Access Manager administrator's user name	sec_master	• configureTAM • reconfigureTAM • unconfigureTAM	Enter the Tivoli Access Manager administration user ID that you created when you configured Tivoli Access Manager. This ID is usually sec_master.
Tivoli Access Manager administrator's user password		• configureTAM • reconfigureTAM • unconfigureTAM	Enter the password that is associated with the Tivoli Access Manager administration user ID.
Tivoli Access Manager security domain	Default	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default.
Embedded Tivoli Access Manager listening port set	8900:8999	• configureTAM • reconfigureTAM	WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999.
Defer	No	• configureTAM • reconfigureTAM • unconfigureTAM	Set this option to *yes* if you want to defer the configuration of the management server until the next restart. Set the option to *no* if you want the configuration of the management server to occur immediately. Managed servers are configured on their next restart.
Force	No	• reconfigureTAM • unconfigureTAM	Set this value to *yes* if you want to ignore errors during the unconfiguration process and allow the entire process to complete. Set the value to *no* if you want errors to stop the unconfiguration process. This option is especially useful if the environment needs to be cleaned up and problems are occurring that do not allow the entire cleanup process to complete successfully.

4. When all information is entered, enter `F` to save the properties or `C` to cancel from the unconfiguration process and discard the entered information.
5. Optional: Synchronize all nodes.
6. Restart all WebSphere Application Server instances for the changes to take effect.

Configuring security auditing using scripting

Security auditing provides tracking and archiving of auditable events. This topic uses the wsadmin tool to enable and administer your security auditing configurations.

About this task

While security authentication and authorization ensures that users must have access to view protected resources, security auditing provides a mechanism to validate the integrity of a security computing environment. Security auditing collects and logs authentication, authorization, system management, security, and audit policy events in audit event records. You can analyze audit event records to determine possible security breaches, threats, attacks, and potential weaknesses in the security configuration of your environment. Enable security auditing in your environment. For example, the following list displays a sample of events to audit:

- Determine the time that a specific user attempted to access a resource.
- View information for successful and unsuccessful attempts to access resources.
- Review changes to resources that were made by a specific user.
- Determine the cause of unsuccessful login attempts.

Use the following task outline to enable and configure security auditing in your environment:

Procedure

1. Enable administrative security in your environment.
2. Configure auditable events. The security auditing configuration provides four default auditable filters. Use this topic to configure filters for additional audit events.
3. Configure audit event factories. The security auditing configuration provides a default event factory. Use this topic to configure additional audit event factories.
4. Configure audit service providers. The security auditing configuration provides a default service provider. Use this topic to configure additional audit service providers.
5. Set the global audit policy. After setting up audit event factories, service providers, and events, use this topic to enable security auditing.

Results

After completing the steps to enable and configure security auditing, the profile of interest audits your security configurations for specific auditable event types.

What to do next

To further configure security auditing, you can:

- Configure audit notifications.
- Encrypt audit records.
- Sign audit records.

Configuring audit service providers using scripting

Before enabling security auditing, use this task to configure audit service providers using the wsadmin tool. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before configuring security audit service providers, enable administrative security in your environment.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

About this task

In order to enable security auditing in your environment, you must configure an audit service provider. The audit service provider writes the audit records and data to the back-end repository associated with the service provide implementation. The security auditing configuration provides a default service provider. Use this topic to customize your security auditing subsystem by creating additional audit service providers.

Use the following steps to configure your security auditing subsystem using the wsadmin tool:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure an audit service provider. You can use the default binary-based audit service provider, or use this step to create a new audit service provider.

   There are binary file-based and third-party audit service providers. In addition to the default binary file-based service provider, you can configure a third-party audit service provider.

   Choose the type of audit service provider to create.

   - Use the createBinaryEmitter command and the following required parameters to create a default audit service provider:

Table 40. Command parameters. This table describes the createBinaryEmitter command parameters.

Parameter	Description	Data Type	Required
-uniqueName	Specifies a unique name that identifies the audit service provider.	String	Yes
-className	Specifies the class implementation of the audit service provider interface.	String	Yes
-fileLocation	Specifies the file location for the audit service provider to write the audit logs.	String	Yes
-auditFilters	Specifies a reference or a group of references to predefined audit filters, using the following format: `reference, reference, reference`	String	Yes

Table 40. Command parameters (continued). This table describes the createBinaryEmitter command parameters.

Parameter	Description	Data Type	Required
-wrapBehavior	Specifies a string representing the customizable behavior for binary audit log wrapping. There are three values for this parameter: WRAP, NOWRAP and SILENT_FAIL If you use the WRAP option, when the maximum logs are reached, the oldest audit log is rewritten; notification is not sent to the auditor. The NOWRAP option does not rewrite over the oldest audit log. It stops the audit service, sends a notification to the SystemOut.log, and quiesces the application server. The SILENT_FAIL option does not rewrite over the oldest audit log. It also stops the audit service, but does allow the WebSphere process to continue. Notifications are not posted in the SystemOut.log.	String	Yes
-maxFileSize	Specifies the maximum size each audit log reaches before the system saves it with a timestamp and creates a new file. Specify the file size in megabytes. If you do not specify this parameter, the system sets the maximum file size to 10 megabytes.	Integer	No
-maxLogs	Specifies the maximum number of audit logs to create before rewriting the oldest log. If you do not specify this parameter, the system allows up to 100 audit logs before overwriting the oldest log.	Integer	No

The following example creates a new audit service provider in your security auditing configuration:

```
AdminTask.createBinaryEmitter('-uniqueName newASP  -wrapBehavior NOWRAP
-className com.ibm.ws.security.audit.BinaryEmitterImpl -fileLocation /AUDIT_logs
-auditFilters "AuditSpecification_1173199825608, AuditSpecification_1173199825609,
AuditSpecification_1173199825610, AuditSpecification_1173199825611"')
```

- Use the createThirdPartyEmitter command to use a third-party audit service provider.

Table 41. Command parameters. Use the following parameters with the createThirdPartyEmitter command:

Parameter	Description	Data Type	Required
-uniqueName	Specifies a unique name that identifies the audit service provider.	String	Yes
-className	Specifies the class implementation of the audit service provider interface.	String	Yes
-eventFormatterClass	Specifies the class that implements how the audit event is formatted for output. If you do not specify this parameter, the system uses the standard text format for output.	String	Yes
-auditFilters	Specifies a reference identifier or a group of reference identifiers to pre-defined audit filters, using the following format: reference, reference, reference.	String	Yes
-customProperties	Specifies any custom properties that might be required to configure a third party audit service provider.	String	No

The following example creates a new third party audit service provider in your security auditing configuration:

```
AdminTask.createThirdPartyEmitter('-uniqueName myAuditServiceProvider -className
com.mycompany.myclass -fileLocation /auditLogs -auditFilters
"AuditSpecification_1173199825608, AuditSpecification_1173199825609,
AuditSpecification_1173199825610, AuditSpecification_1173199825611"')
```

3. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

What to do next

Enable security auditing in your environment.

Configuring audit event factories using scripting

Before enabling security auditing, use this task to configure audit event factories using the wsadmin tool. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before configuring security auditing event factories, enable administrative security in your environment.

About this task

In order to enable security auditing in your environment, you must configure an audit event factory. The audit event factory gathers the data that is associated with security events. The security auditing configuration provides a default event factory. Use this topic to customize your security auditing subsystem by creating additional audit event factories.

Use the following steps to configure your security auditing subsystem using the wsadmin tool:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure event filters. You can use the default event filters or use this step to create additional filters to customize your security auditing configuration.

Table 42. Event filters in the audit.xml file. The application server provides the following event filters by default in the `audit.xml` template file:

Event Name	Outcome of event
SECURITY_AUTHN	SUCCESS
SECURITY_AUTHN	DENIED
SECURITY_RESOURCE_ACCESS	SUCCESS
SECURITY_AUTHN	REDIRECT

You can configure additional audit event types to track and archive various events. Use the following command to list all supported auditable events:

```
print AdminTask.getSupportedAuditEvents()
```

Use the createAuditFilter command with the -eventType and -outcome parameters to enable one or multiple audit events and outcomes. You can specify multiple event types and multiple outcomes separated by a comma with one command invocation. The following list describes each valid auditable event that you can specify with the -eventType parameter:

Table 43. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.

Table 43. Event types (continued). Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

Important: The following security audit event types are not used in this release of WebSphere Application Server:

- SECURITY_MGMT_KEY
- SECURITY_RUNTIME_KEY
- SECURITY_MGMT_PROVISIONING
- SECURITY_MGMT_REGISTRY
- SECURITY_RUNTIME

For each audit event type, you must specify an outcome. Valid outcomes include SUCCESS, FAILURE, REDIRECT, ERROR, DENIED, WARNING, and INFO. The following command example creates an audit filter to log users who receive an error when modifying credentials:

```
AdminTask.createAuditFilter('-name uniqueFilterName -eventType
SECURITY_AUTHN_CREDS_MODIFY,SECURITY_AUTHN_DELEGATION -outcome ERROR,REDIRECT')
```

3. Create an audit event factory. You can use the default audit event factory or use this step to create a new audit event factory.

   Use the createAuditEventFactory command to create an audit event factory in your security configuration. You can use the default implementation of the audit event factory or use a third-party implementation. To configure a third-party implementation, use the optional -customProperties parameter to specify any properties necessary to configure the audit event factory implementation.

Table 44. Required parameters. Specify the following required parameters with the createAuditEventFactory to configure your audit event factory:

Parameter	Description	Data type	Required
-uniqueName	Specifies a unique name that identifies the audit event factory.	String	Yes
-className	Specifies the class implementation of the audit event factory interface.	String	Yes
-auditFilters	Specifies a reference or a group of references to predefined audit filters, using the following format: `"reference, reference, reference"`	String	Yes
-provider	Specifies a reference to a predefined audit service provider implementation.	String	Yes
-customProperties	Specifies a comma (,) separated list of custom property pairs to add to the security object in the following format: `attribute=value,attribute=value`	String	No

The following sample command creates an enables an audit event factory:

```
AdminTask.createAuditEventFactory('-uniqueName eventFactory1 -className
com.ibm.ws.security.audit.AuditEventFactoryImpl -auditFilters
"AuditSpecification_1173199825608, AuditSpecification_1173199825609, AuditSpecification_1173199825610,
AuditSpecification_1173199825611" -provider newASP')
```

4. Save your configuration changes.

Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

What to do next

Configure the audit service provider.

Configuring auditable events using scripting

Before enabling security auditing, use this task to configure event filters using the wsadmin tool. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before configuring security auditing filters, enable administrative security in your environment.

About this task

Before configuring an audit event factory and audit service provider, configure event filters. The audit service provider writes audit records to the back end repository associated with the provider implementation. The audit event factory generates security events. Event filters specify which event types and outcomes the system audits and records. Each event type has up to seven possible outcomes, including success, failure, denied, error, warning, info, and redirect. The security auditing configuration provides four default filters. Use this topic to customize your security auditing subsystem by creating additional audit event filters.

Use the following steps to configure your security auditing subsystem using the wsadmin tool:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure event filters. You can use the default event filters or use this step to create additional filters to customize your security auditing configuration.

Table 45. Provided event filters. The application server provides the following event filters by default in the `audit.xml` template file:

Event Name	Outcome of event
SECURITY_AUTHN	SUCCESS
SECURITY_AUTHN	DENIED
SECURITY_RESOURCE_ACCESS	SUCCESS
SECURITY_AUTHN	REDIRECT

You can configure additional audit event types to capture various events. Use the following command to list all supported auditable events:

```
print AdminTask.getSupportedAuditEvents()
```

Use the createAuditFilter command with the -name, -eventType, and -outcome parameters to enable one or multiple audit events and outcomes. You can specify multiple event types and multiple outcomes separated by a comma with one command invocation.

Table 46. Event types. The following list describes each valid auditable event that you can specify with the -eventType parameter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies

Table 46. Event types (continued). The following list describes each valid auditable event that you can specify with the -eventType parameter:

Event name	Description
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.

For each audit event type, you must specify an outcome. Valid outcomes include SUCCESS, FAILURE, REDIRECT, ERROR, DENIED, WARNING, and INFO. The following command example creates an audit filter to log users who receive an error when modifying credentials:

```
AdminTask.createAuditFilter('-name myUniqueName -eventType SECURITY_AUTHN_CREDS_MODIFY,SECURITY_AUTHN_DELEGATION -outcome ERROR,REDIRECT')
```

3. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

What to do next

Enable security auditing in your environment.

Enabling security auditing using scripting

Use this task to enable and configure security auditing in your environment with the wsadmin tool. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before enabling security auditing, enable administrative security in your environment.

If you previously configured security auditing and do not want to modify configuration settings, use the enableAudit and disableAudit commands to start and stop security auditing. After enabling or disabling security auditing, restart the server to apply the configuration changes.

About this task

Security auditing ensures the integrity of a security computing environment. Security auditing collects and logs authentication, authorization, system management, security, and audit policy events in audit event records. You can analyze audit event records to determine possible security breaches, threats, attacks, and potential weaknesses in the security configuration of your environment.

Use the following steps to enable and configure security auditing in your environment:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Verify that the security auditing subsystem is configured.

   To enable security auditing, you must configure event filters, an audit emitter, and an audit event factory. Event filters specify which event types the system audits and records, and the outcome of the event. The audit service provider writes the audit records to the backend repository that is associated with the implementation. The audit event factory generates security events.

   By default, the security auditing system includes one audit service provider and one audit event factory.

The audit command groups provide several commands to query for event filters, audit emitters, event factories, and their respective configuration attributes. Use the audit command reference to use specific query commands. The following example commands query your security auditing configuration at a high level.

- Use the getAuditFilters command to display a list of references to all audit filters defined in your configuration, as the following example demonstrates:

```
AdminTask.getAuditFilters()
```

- Use the listAuditEmitters command to display a list of all audit emitters in your configuration, as the following example demonstrates:

```
AdminTask.listAuditEmitters()
```

- Use the listAuditEventFactories command to display a list of all audit event factories in your configuration, as the following example demonstrates:

```
AdminTask.listAuditEventFactories()
```

3. Enable security auditing in your environment. Use the modifyAuditPolicy command to enable security auditing in your environment.

Table 47. Command parameters. Use the following optional parameters for the modifyAuditPolicy command to customize your security auditing configuration:

Parameter	Description	Data type	Required
-auditEnabled	Specifies whether to enable security auditing.	Boolean	No
-auditPolicy	Specifies the behavior of the server process if the audit subsystem fails. Valid values are: `WARN`, `NOWARN` and `FATAL`. The `WARN` setting notifies the auditor when an error occurs and ceases auditing when an error occurs in the audit sub-system, but continues to run the application server process. The `NOWARN` setting does not notify the auditor when an error occurs and ceases auditing, but continues to run the application server process. The `FATAL` setting notifies the auditor of the error and stops the application server process. By default, the command assigns the `NOWARN` setting.	String	No
-auditorId	Specifies the ID of the user to assign to the auditor role.	String	No
-auditorPwd	Specifies the password for the auditor role.	String	No
-sign	Specifies whether to sign audit records. By default, the security auditing system does not sign audit records. You must configure the signing of audit records before you can specify this parameter.	Boolean	No
-encrypt	Specifies whether to encrypt audit records. By default, the security auditing system does not encrypt audit records. You must configure encryption for audit records before you can specify this parameter.	Boolean	No
-verbose	Specifies whether to capture verbose audit data. By default, the security auditing system does not capture verbose audit data.	Boolean	No
-encryptionCert	Specifies the reference ID of the certificate to use for encryption. Specify this parameter if you set the -encrypt parameter to `true`.	String	No

The following example command enables security auditing, and identifies the primary auditor by assigning a user and password.

```
AdminTask.modifyAuditPolicy('-auditEnabled true -auditorId securityAdmin -auditorPwd security4you')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

5. Restart the server.

Results

After completing the steps to enable and configure security auditing, the profile of interest audits your security configurations for specific auditable event types.

What to do next

After you configure the audit policy for the first time, use the enableAudit and disableAudit commands to turn the security auditing system on and off. The system maintains the settings that you define with the modifyAuditPolicy command when you enable and disable the security auditing system.

Note: You must restart the server to apply the configuration changes.

Configuring security audit notifications using scripting

Configure the security auditing system to send email notifications to a distribution list, system log, or both a distribution list and a system log if a failure occurs in the audit subsystem. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before configuring a notification object in the audit.xml configuration file, verify that you set up a security auditing subsystem and configured the security auditing policy.

About this task

You can configure the security auditing system to notify a specific person or group when a failure occurs in the audit subsystem. Use the following steps to enable security auditing email notifications, set the format of notification email, and secure email:

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Customize and enable security auditing email notifications.

Table 48. Command parameters. Use the createAuditNotification command and the following parameters to configure notifications:

Parameter	Description	Data Types	Required
-notificationName	Specifies a unique name to assign the audit notification object in the audit.xml file.	String	Yes

Table 48. Command parameters (continued). Use the createAuditNotification command and the following parameters to configure notifications:

Parameter	Description	Data Types	Required
-logToSystemOut	Specifies whether to log the notification to the SystemOut.log file.	Boolean	Yes
-sendEmail	Specifies whether to email notifications.	Boolean	Yes
-emailList	Specifies the email address or email distribution list to email notifications. The format for this parameter is: `admin@company.com(smtp-server.mycompany.com)`	String	No
-emailFormat	Specifies whether to send the email be `HTML` or `TEXT` format.	String	No

To create the audit notification object, you must specify the -notificationName, -logToSystemOut, and -sendEmail parameters, as the following example demonstrates:

```
AdminTask.createAuditNotification('-notificationName defaultEmailNotification
-logToSystemOut true -sendEmail true -emailList administrator@mycompany.com(smtp-server.mycompany.com)
-emailFormat HTML')
```

3. Create an audit notification monitor object.

   Create an audit notification monitor object to monitor the security auditing subsystem for possible failure.

Table 49. Command parameters. Use the createAuditNotificationMonitor command and the following parameters to create a monitor object for the security auditing system:

Parameter	Description	Data Types	Required
-notificationName	Specifies a unique name to assign the audit notification object in the audit.xml file.	String	Yes
-logToSystemOut	Specifies whether to log the notification to the SystemOut.log file.	Boolean	Yes
-sendEmail	Specifies whether to email notifications.	Boolean	Yes
-emailList	Specifies the email address or email distribution list to email notifications. The format for this parameter is: `admin@company.com(smtp-server.mycompany.com)`	String	No
-emailFormat	Specifies whether to send the email be `HTML` or `TEXT` format.	String	No

To create the audit notification monitor object, you must specify the -notificationName, -logToSystemOut, and -sendEmail parameters, as the following example demonstrates:

```
AdminTask.createAuditNotificationMonitor('-notificationName defaultEmailNotification
-logToSystemOut true -sendEmail true -emailList administrator@mycompany.com(smtp-server.mycompany.com)
-emailFormat HTML')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

Results

The security auditing system notifies the specified recipients if a failure occurs in the security auditing system.

What to do next

Use the modifyAuditNotification command and the Audit Notification Commands command group for the AdminTask object to manage your notification configuration.

Encrypting security audit data using scripting

You can use the wsadmin tool to configure the security auditing system to encrypt security audit records. Security auditing provides tracking and archiving of auditable events.

Before you begin

Before configuring encryption, set up your security auditing subsystem. You can enable security auditing before or after completing the steps in this topic.

Verify that you have the appropriate administrative role. To complete this topic, you must have the auditor administrative role. If you are importing a certificate from a keystore that exists in the security.xml file, you must have the auditor and administrator administrative roles.

About this task

When configuring encryption, the auditor can select one of the following choices:

- Allow the application server to automatically generate a certificate or use an existing self-signed certificate generated by the auditor.
- Use an existing keystore to store this certificate, or create a new keystore to store this certificate.

Note: To ensure that there is a separation of privileges between the administrator role and the auditor role, the auditor can create a self-signed certificate outside of the application server process and maintain the private key of that certificate.

Use the following task steps to encrypt security audit data:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure encryption settings for security audit data.

   Use the createAuditEncryptionConfig command and the following parameters to create the audit encryption model to encrypt your audit records. You must specify the -enableAuditEncryption, -certAlias, and -encryptionKeyStoreRef parameters, and either the -autogenCert or -importCert parameters.

Table 50. Command parameters. This table describes the createAuditEncryptionConfig command and its parameters:

Parameter	Description	Data Type	Required
-enableAuditEncryption	Specifies whether to encrypt audit records. This parameter modifies your audit policy configuration.	Boolean	Yes
-certAlias	Specifies the alias name that identifies the generated or imported certificate.	String	Yes
-encryptionKeyStoreRef	Specifies the reference ID of the keystore to import the certificate to.	String	Yes
-autogenCert	Specifies whether to automatically generate the certificate used to encrypt the audit records. You must specify either this parameter or the -importCert parameter, but you cannot specify both.	Boolean	No

Table 50. Command parameters (continued). This table describes the createAuditEncryptionConfig command and its parameters:

Parameter	Description	Data Type	Required
-importCert	Specifies whether to import an existing certificate to encrypt the audit records. You must specify either this parameter or the -autogenCert parameter, but you cannot specify both.	Boolean	No
-certKeyFileName	Specifies the unique name of the key file from which the certificate is imported.	String	No
-certKeyFilePath	Specifies the key file location from which the certificate is imported.	String	No
-certKeyFileType	Specifies the key file type from which the certificate is imported.	String	No
-certKeyFilePassword	Specifies the key file password from which the certificate is imported.	String	No
-certAliasToImport	Specifies the alias from which the certificate is imported.	String	No

The following command example configures encryption and supports the system to automatically generate the certificate:

```
AdminTask.createAuditEncryptionConfig('-enableAuditEncryption true -certAlias auditCertificate
-autogenCert true -encryptionKeyStoreRef auditKeyStore')
```

The following command example configures encryption and imports a certificate:

```
AdminTask.createAuditEncryptionConfig('-enableAuditEncryption true -certAlias auditCertificate
-importCert true -certKeyFileName MyServerKeyFile.p12 -certKeyFilePath
install_root/etc/MyServerKeyFile.p12 -certKeyFileType PKCS12 -certKeyFilePassword password4key
-certAliasToImport defaultCertificate -encryptionKeyStoreRef auditKeyStore')
```

3. You must restart the server to apply configuration changes.

Results

Encryption is configured for security audit data. If you set the -enableAuditEncryption parameter to `true`, then your security auditing system encrypts security audit data when security auditing is enabled.

What to do next

After you configure the encryption model for the first time, then you may use the enableAuditEncryption and disableAuditEncryption commands to turn encryption on and off.

The following example uses the enableAuditEncryption command to turn on encryption:

```
AdminTask.enableAuditEncryption()
```

The following example uses the disableAuditEncryption command to turn off encryption:

```
AdminTask.disableAuditEncryption()
```

Signing security audit data using scripting

You can use the wsadmin tool to configure the security auditing system to sign security audit records. Security auditing provides tracking and archiving of auditable events.

Before you begin

Verify that you have the appropriate administrative role. To complete this topic, you must have the auditor and administrator administrative roles.

About this task

When configuring the signing of audit data, the auditor can choose between the following options:

- Allow the application server to automatically generate a certificate.
- Use an existing self-signed certificate that the auditor previously generated.
- Use the same self-signed certificate as the system uses to encrypt the audit records.
- Use an existing keystore to store this certificate.
- Create a new keystore to store this certificate.
- Use an existing self-signed certificate in an existing keystore.

Use the following task steps to configure the signing of security audit data:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. See the Starting the wsadmin scripting client article for more information.
2. Configure signing settings for security audit data.

   Use the createAuditSigningConfig command to create the signing model to sign your audit records.

   You can import the certificate from an existing key file name that contains that certificate, automatically generate the certificate, or use the same certificate as used to encrypt the audit records. The signing keystore must exist in the `security.xml` file. The system updates this keystore with the certificate to use to sign the audit records.

Table 51. Command parameters. Use the parameters in the following table with the createAuditSigningConfig command. You must specify the -enableAuditSigning, -certAlias, and -signingKeyStoreRef parameters.

Parameter	Description	Data Type	Required
-enableAuditSigning	Specifies whether to sign audit records. This parameter modifies your audit policy configuration.	Boolean	Yes
-certAlias	Specifies the alias name that identifies the generated or imported certificate.	String	Yes
-signingKeyStoreRef	Specifies the reference ID of the keystore to import the certificate to.	String	Yes
-useEncryptionCert	Specifies whether to use the same certificate for encryption and signing. You must specify the -useEncryptionCert, -autogenCert, or -importCert parameter.	Boolean	No
-autogenCert	Specifies whether to automatically generate the certificate used to sign the audit records. You must specify the -useEncryptionCert, -autogenCert, or -importCert parameter.	Boolean	No
-importCert	Specifies whether to import an existing certificate to sign the audit records. You must specify the -useEncryptionCert, -autogenCert, or -importCert parameter.	Boolean	No
-certKeyFileName	Specifies the unique name of the key file for the certificate to import.	String	No
-certKeyFilePath	Specifies the key file location for the certificate to import.	String	No
-certKeyFileType	Specifies the key file type for the certificate to import.	String	No
-certKeyFilePassword	Specifies the key file password for the certificate to import.	String	No
-certAliasToImport	Specifies the alias of the certificate to import.	String	No

The following command example configures signing and allows the system to automatically generate the certificate:

```
AdminTask.createAuditSigningConfig('-enableAuditSigning true -certAlias auditSigningCert
-autogenCert true -signingKeyStoreRef Ref_Id_of_KeyStoreInSecurityXML')
```

The following command example configures signing and imports a certificate:

```
AdminTask.createAuditSigningConfig('-enableAuditSigning true -certAlias auditSigningCert
-importCert true -certKeyFileName MyServerKeyFile.p12 -certKeyFilePath install_root/etc/MyServerKeyFile.p12
-certKeyFileType PKCS12 -certKeyFilePassword password4key -certAliasToImport defaultCertificate
-signingKeyStoreRef Ref_Id_of_KeyStoreInSecurityXML')
```

The following command example uses the same certificate for signing and encryption:

```
AdminTask.createAuditSigningConfig('-enableAuditSigning true -certAlias auditSigningCert
-useEncryptionCert true -signingKeyStoreRef Ref_Id_of_KeyStoreInSecurityXML')
```

3. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

4. Restart the server to apply the configuration changes.

Results

Signing is configured for your security audit data. If you set the -enableAuditSigning parameter to `true`, your security auditing system signs security audit data when security auditing is enabled.

What to do next

Once you configure the signing model for the first time, use the enableAuditSigning and disableAuditSigning commands to quickly turn signing on and off. The following example uses the enableAuditSigning command to turn signing on:

```
AdminTask.enableAuditSigning()
```

The following example uses the disableAuditSigning command to turn signing off:

```
AdminTask.disableAuditSigning()
```

AuditKeyStoreCommands command group for the AdminTask object

You can use the Jython scripting language to configure the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditKeyStoreCommands group to configure audit keystores in the security auditing system.

Use the following commands to manage audit key stores in the `audit.xml` configuration file:

- createAuditKeyStore
- deleteAuditKeyStore
- getAuditKeyStoreInfo
- listAuditKeyStores
- modifyAuditKeyStore

createAuditKeyStore

Creates a keystore in the `audit.xml` file. The system uses this keystore to encrypt audit records.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

-keyStoreType
Specifies a valid keystore type. The default keystore type is `PKCS12`. (String, required)

-keyStoreLocation
Specifies the location where the system creates the keystore. (String, required)

-keyStorePassword
Specifies the password for the keystore. (String, required)

-keyStorePasswordVerify
Verifies the password for the keystore. (String, required)

Optional parameters

-keyStoreProvider
Specifies a provider for the keystore. (String, optional)

-keyStoreIsFileBased
Specifies if the keystore is file-based. The default is `true`. (Boolean, optional)

-keyStoreHostList
Specifies the host list for the keystore. (String, optional)

-keyStoreInitAtStartup
Specifies whether the system initializes the keystore on startup. The default is `false`. (Boolean, optional)

-keyStoreReadOnly
Specifies whether the keystore is read-only or not. Default is `false`. (Boolean, optional)

-keyStoreStashFile
Specifies whether the keystore needs a stash file. (Boolean, optional)

-enableCryptoOperations
Specifies whether the keystore is an acceleration keystore. False default. (Boolean, optional)

-scopeName
Specifies the scope for the keystore. (String, optional)

-keyStoreDescription
Specifies a description for the keystore. (String, optional)

Return value

The command returns the ID of the new keystore, as the following example displays:

```
KeyStore_1173199825578
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditKeyStore('-keyStoreName mynewkeystore -keyStoreLocation
c:\install_root\appserver\profiles\AppSrv01\config\cells -keyStorePassword
myPwd -keyStorePasswordVerify myPwd -keyStoreProvider IBMJCE -scopeName (cell):Node04Cell')
```

- Using Jython list:

```
AdminTask.createAuditKeyStore(['-keyStoreName', 'mynewkeystore', '-keyStoreLocation',
'c:\install_root\appserver\profiles\AppSrv01\config\cells', '-keyStorePassword',
'myPwd', '-keyStorePasswordVerify', 'myPwd', '-keyStoreProvider', 'IBMJCE',
'-scopeName', '(cell):Node04Cell'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditKeyStore('-interactive')
```

deleteAuditKeyStore

The deleteAuditKeyStore command removes the reference to an audit keystore from the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the name of the keystore. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the keystore. (String, optional)

-removeKeyStoreFile
Specifies whether to remove the keystore from the configuration. Specify this parameter if the keystore of interest is not in use. (Boolean, optional)

Return value

The command returns a value of `true` if the system successfully removes the reference to the keystore from the `audit.xml` configuration file.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditKeyStore('-keyStoreName AuditDefaultKeyStore -scopeName
(cell):Node04Cell -removeKeyStoreFile false')
```

- Using Jython list:

```
AdminTask.deleteAuditKeyStore(['-keyStoreName', 'AuditDefaultKeyStore', '-scopeName',
'(cell):Node04Cell', '-removeKeyStoreFile', 'false'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditKeyStore('-interactive')
```

getAuditKeyStoreInfo

The getAuditKeyStoreInfo command returns a list of attributes for the keystore that the system uses to encrypt audit records.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name to identify the keystore. (String, required)

Optional parameters

-scopeName
Specifies the management scope of the keystore. (String, optional)

Return value

The command returns a list of attributes for the keystore, as the following sample output displays:

```
{{location ${CONFIG_ROOT}/audittrust.p12}
{password *****}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#KeyStore_1173199825578}
{_Websphere_Config_Data_Version {}}
{useForAcceleration false}
{slot 0}
{type PKCS12}
{additionalKeyStoreAttrs {}}
{fileBased true}
{_Websphere_Config_Data_Type KeyStore}
{customProviderClass {}}
{hostList {}}
{createStashFileForCMS false}
{description {keyStore description}}
{readOnly false}
{initializeAtStartup true}
{managementScope (cells/Node04Cell|audit.xml#ManagementScope_1173199825608)}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditKeyStoreInfo('-keyStoreName AuditDefaultKeyStore')
```

- Using Jython list:

```
AdminTask.getAuditKeyStoreInfo(['-keyStoreName', 'AuditDefaultKeyStore'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditKeyStoreInfo('-interactive')
```

listAuditKeyStores

The listAuditKeyStores command lists the attributes for the audit keystores within a specific management scope or for all audit keystores.

The user must have the monitor administrative role to run this command.

Target object

None.

Optional parameters

-scopeName
Specifies the management scope associated with the keystores of interest. (String, optional)

-all
Specifies whether to list all keystores. When the -all parameter is set as `true`, it overrides the -scopeName parameter. (Boolean, optional)

Return value

The command returns a list of attributes for the scope of interest, as the following sample output displays:

```
{{location ${CONFIG_ROOT}/audittrust.p12}
{password *****}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#KeyStore_1173199825578}
{_Websphere_Config_Data_Version {}}
{useForAcceleration false}
{slot 0}
```

```
{type PKCS12}
{additionalKeyStoreAttrs {}}
{fileBased true}
{_Websphere_Config_Data_Type KeyStore}
{customProviderClass {}}
{hostList {}}
{keyStoreRef KeyStore_1173199825578}
{createStashFileForCMS false}
{description {keyStore description}}
{managementScope (cells/Node04Cell|audit.xml#ManagementScope_1173199825608)}
{readOnly false}
{initializeAtStartup true}
{usage {}}
{provider IBMJCE}{name AuditDefaultKeyStore}}
{{location c:\install_root\appserver\profiles\AppSrv01\config\cells}
{password *****}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#KeyStore_1184700968484}
{_Websphere_Config_Data_Version {}}
{useForAcceleration false}
{slot 0}
{type PKCS12}
{additionalKeyStoreAttrs {}}
{fileBased true}
{_Websphere_Config_Data_Type KeyStore}
{customProviderClass {}}
{hostList {}}
{keyStoreRef KeyStore_1184700968484}
{createStashFileForCMS false}
{description {}}
{managementScope {}}
{readOnly false}
{initializeAtStartup false}
{usage {}}
{provider IBMJCE}
{name mykeystore}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditKeyStores('-scopeName (cell):Node04Cell')
```

- Using Jython list:

```
AdminTask.listAuditKeyStores(['-scopeName', '(cell):Node04Cell'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditKeyStores('-interactive')
```

modifyAuditKeyStore

The modifyAuditKeyStore command modifies the keystore reference in the `audit.xml` file. The command edits keystore that encrypts audit records.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

Optional parameters

-scopeName
Specifies the scope name of this keystore. (String, optional)

-keyStoreType
Specifies valid keystore type. (String, optional)

-keyStoreLocation
Specifies the location where the system creates the keystore. (String, optional)

-keyStorePassword
Specifies the password for this keystore. (String, optional)

-keyStoreIsFileBased
Specifies whether the keystore is file based. (Boolean, optional)

-keyStoreInitAtStartup
Specifies whether the system should initialize the keystore at startup. (Boolean, optional)

-keyStoreReadOnly
Specifies whether the keystore is read-only or editable. (Boolean, optional)

-keyStoreDescription
Specifies a description for the keystore. (String, optional)

Return value

The command returns a value of `true` if the system successfully modifies the keystore.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditKeyStore('-keyStoreName AuditDefaultKeyStore -scopeName
(cell):Node04Cell -keyStoreType PKCS12 -keyStoreLocation
c:\install_root\appserver\profiles\AppSrv01\config\cells\Node04Cell\auditttrust.p12
-keyStorePassword myPwd')
```

- Using Jython list:

```
AdminTask.modifyAuditKeyStore(['-keyStoreName', 'AuditDefaultKeyStore', '-scopeName',
'(cell):Node04Cell', '-keyStoreType', 'PKCS12', '-keyStoreLocation',
'c:\install_root\appserver\profiles\AppSrv01\config\cells\Node04Cell\auditttrust.p12',
'-keyStorePassword', 'myPwd'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditKeyStore('-interactive')
```

AuditEmitterCommands for the AdminTask object

You can use the Jython scripting language to configure audit service providers with the wsadmin tool. Use the commands and parameters in the AuditEmitterCommands group to create, manage, and remove audit service providers from your security auditing system configuration.

Use the following commands to configure audit service providers:

- “modifyAuditEmitter” on page 309
- “setAuditEmitterFilters” on page 310

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

createBinaryEmitter

The createBinaryEmitter command creates an entry in the `audit.xml` file to reference the configuration of the binary file emitter implementation of the audit service provider interface.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies a name to uniquely identify this implementation of the audit service provider interface. (String, required)

-className
Specifies the class that implements the audit service provider interface. (String, required)

-fileLocation
Specifies the location where the system writes the audit logs. (String, required)

-auditFilters
Specifies a reference or a group of references to predefined audit filters. Use the following format to specify multiple references: `reference,reference,reference` (String, required)

-wrapBehavior
Specifies a string representing the customizable behavior for binary audit log wrapping. (String, required).

There are three values for this parameter: WRAP, NOWRAP and SILENT_FAIL

If you use the WRAP option, when the maximum logs are reached, the oldest audit log is rewritten; notification is not sent to the auditor.

The NOWRAP option does not rewrite over the oldest audit log. It stops the audit service, sends a notification to the SystemOut.log, and quiesces the application server.

The SILENT_FAIL option does not rewrite over the oldest audit log. It also stops the audit service, but does allow the WebSphere process to continue. Notifications are not posted in the SystemOut.log.

Note: If you use the NOWRAP or SILENT_FAIL options, when the server is stopped as a result of the logs being maxed-out, a stopserver is performed, or because the server abends in some way, you must archive the binary audit logs before you restart the server.

Optional parameters

-eventFormatterClass
Specifies the class that implements how the system formats the audit event for output. If you want to use the default audit service provider, do not specify this parameter. (String, optional)

-maxFileSize
Specifies the maximum size that each log reaches before the system saves the audit log with a timestamp. Specify the size in megabytes. The default value is 10 MB. (Integer, optional)

-maxLogs
Specifies the maximum number of log files to create before the system rewrites the oldest audit log. The default value is 100 logs. (Integer, optional)

Return value

The command returns the shortened reference ID for the audit service provider, as the following sample output displays:

```
AuditServiceProvider_1184686384968
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createBinaryEmitter('-uniqueName mybinaryemitter -className
com.ibm.ws.security.audit.BinaryEmitterImpl -fileLocation
c:\wasinstall\appserver\profiles\AppSrv01\logs\server1 -maxFileSize 20 -maxLogs
100 –wrapBehavior NOWRAP -auditFilters AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.createBinaryEmitter(['-uniqueName', 'mybinaryemitter', '-className',
'com.ibm.ws.security.audit.BinaryEmitterImpl', '-fileLocation',
'c:\wasinstall\appserver\profiles\AppSrv01\logs\server1', '-maxFileSize',
'20', '-maxLogs', '100', '–wrapBehavior', 'NOWRAP', '-auditFilters',
'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createBinaryEmitter('-interactive')
```

createSMFEmitter

The createSMFEmitter command creates an entry in the `audit.xml` file to reference the configuration of an SMF implementation of the audit service provider interface. The encryption and signing of audit records is not supported for SMF implementations.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies a name to uniquely identify this implementation of the audit service provider interface. (String, required)

-auditFilters
Specifies a reference or a group of references to predefined audit filters. Use the following format to specify multiple references: `reference,reference,reference` (String, required)

Return value

The command returns the shortened reference ID for the audit service provider, as the following sample output displays:

```
AuditServiceProvider_1184686384968
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createSMFEmitter('-uniqueName mySMFEmitter -auditFilters
AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.createSMFEmitter(['-uniqueName', 'mySMFEmitter', '-auditFilters',
'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createSMFEmitter('-interactive')
```

createThirdPartyEmitter

The createThirdPartyEmitter command creates an entry in the `audit.xml` configuration file to reference the configuration of a third party emitter implementation of the audit service provider interface. The encryption and signing of audit records is not supported for third party implementations.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies a name to uniquely identify this implementation of the audit service provider interface. (String, required)

-className
Specifies the class that implements the audit service provider interface. (String, required)

-auditFilters
Specifies a reference or a group of references to predefined audit filters. Use the following format to specify multiple references: `reference,reference,reference` (String, required)

Optional parameters

-eventFormatterClass
Specifies the class that implements how the system formats the audit event for output. (String, optional)

-customProperties
Specifies any custom properties that the system might need to configure the third party implementation of the audit service provider. Use the following format to specify the custom properties: `name=value,name=value` (String, optional)

Return value

The command returns the shortened reference ID to the audit service provider, as the following example output displays:

```
AuditServiceProvider_1184686638218
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createThirdPartyEmitter('-uniqueName myThirdPartyEmitter -className
com.mycompany.myemitterclass -eventFormatterClass com.mycompany.myeventformatterclass
-auditFilters AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.createThirdPartyEmitter(['-uniqueName', 'myThirdPartyEmitter', '-className',
'com.mycompany.myemitterclass', '-eventFormatterClass', 'com.mycompany.myeventformatterclass',
'-auditFilters', 'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createThirdPartyEmitter('-interactive')
```

deleteAuditEmitterByRef

The deleteAuditEmitterByRef command deletes the audit service provider implementation that the system references with the reference id. If an event factory is using the audit service provider, the system generates an error that indicates that the system cannot remove the audit service provider.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies the reference identifier of the audit service provider implementation to delete. (String, required)

Return value

The command returns a value of `true` if the system successfully removes the audit service provider.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditEmitterByRef('–emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.deleteAuditEmitterByRef(['–emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditEmitterByRef('-interactive')
```

deleteAuditEmitterByName

The deleteAuditEmitterByName command deletes the audit service provider implementation that the system references with the unique name. If an event factory is using the audit service provider, the system generates an error that indicates that the system cannot remove the audit service provider.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies the name that uniquely identifies this implementation of the audit service provider interface to delete. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit service provider implementation.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditEmitterByName('-uniqueName mybinaryemitter')
```

- Using Jython list:

```
AdminTask.deleteAuditEmitterByName(['-uniqueName', 'mybinaryemitter'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditEmitterByName('-interactive')
```

getAuditEmitter

The getAuditEmitter command returns the attributes for the audit service provider of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a audit service provider implementation. (String, required)

Return value

The command returns an attribute list for the audit service provider specified by the -emitterRef parameter, as the following example output displays:

```
{{auditSpecifications myfilter(cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1184598886859)}
{name auditServiceProviderImpl_1}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditServiceProvider_1173199825608}
{maxFileSize 1}
{_Websphere_Config_Data_Type AuditServiceProvider}
{fileLocation ${PROFILE_ROOT}/logs/server1}
{className com.ibm.ws.security.audit.BinaryEmitterImpl}
{properties {}}
{eventFormatterClass {}}
{maxLogs 100}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEmitter('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getAuditEmitter(['-emitterRef AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEmitterClass('-interactive')
```

getBinaryFileLocation

The getBinaryFileLocation command returns the file location of the binary file audit logs.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a binary file audit service provider implementation. (String, required)

Return value

The command returns the file path of the audit log, as the following example displays:

```
$profile_root/logs/server1
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getBinaryFileLocation('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getBinaryFileLocation(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getBinaryFileLocation('-interactive')
```

getAuditEmitterFilters

The getAuditEmitterFilters command returns a list of defined filters for the audit service provider implementation of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies the audit service provider implementation of interest. You can specify a reference to the service provider object. (String, required)

Return value

The command returns a list of defined filters in a shortened format, as the following example output displays:

```
AUTHN:SUCCESS,AUTHN:INFO,AUTHZ:SUCCESS,AUTHZ:INFO
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEmitterFilters('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getAuditEmitterFilters(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEmitterFilters('-interactive')
```

getBinaryFileSize

The getBinaryFileSize command returns the maximum file size of the binary audit log that is defined for the audit service provider of interest in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a binary file audit service provider implementation. (String, required)

Return value

The command returns the integer value of the maximum file size in megabytes.

Batch mode example usage

- Using Jython string:

```
AdminTask.getBinaryFileSize('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getBinaryFileSize(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getBinaryFileSize('-interactive')
```

getEmitterClass

The getEmitterClass command returns the class name of the audit service provider emitter implementation.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a audit service provider implementation. (String, required)

Return value

The command returns the class name of the audit service provider implementation.

Batch mode example usage

- Using Jython string:

```
AdminTask.getEmitterClass('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getEmitterClass(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEmitterClass('-interactive')
```

getEmitterUniqueId

The getEmitterUniqueId command returns the unique identifier of the audit service provider implementation.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a service provider implementation. (String, required)

Return value

The command returns the unique ID of the audit service provider of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.getEmitterUniqueId('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getEmitterUniqueId(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEmitterUniqueId('-interactive')
```

getMaxNumBinaryLogs

The getMaxNumBinaryLogs command returns the maximum number of binary audit logs that is defined for the audit service provider of interest in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a binary file audit service provider implementation. (String, required)

Return value

The command returns the integer value that represents the maximum number of binary audit logs in the configuration.

Batch mode example usage

- Using Jython string:

```
AdminTaskgetMaxNumBinaryLogs('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTaskgetMaxNumBinaryLogs(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getMaxNumBinaryLogs('-interactive')
```

listAuditEmitters

The listAuditEmitters command returns a list of configured audit service provider implementation objects and the corresponding attributes.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns an array list of audit service provider implementation objects and attributes, as the following example output displays:

```
{{auditSpecifications myfilter(cells/CHEYENNENode04Cell|audit.xml#AuditSpecifica
tion_1184598886859)}
{name auditServiceProviderImpl_1}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditServiceProvid
er_1173199825608}
{maxFileSize 1}
{_Websphere_Config_Data_Type AuditServiceProvider}
{fileLocation ${PROFILE_ROOT}/logs/server1}
{className com.ibm.ws.security.audit.BinaryEmitterImpl}
{properties {}}
{auditSpecRef1 AuditSpecification_1184598886859}
{eventFormatterClass {}}
{maxLogs 100}
{emitterRef AuditServiceProvider_1173199825608}}
{{auditSpecifications DefaultAuditSpecification_1(cells/CHEYENNENode04Cell|audit
.xml#AuditSpecification_1173199825608)}
{name mythirdpartyemitter}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditServiceProvid
er_1184686638218}
{maxFileSize 0}
{_Websphere_Config_Data_Type AuditServiceProvider}
{fileLocation {}}
{className com.mycompany.myemitterclass}
{properties {}}
{auditSpecRef1 AuditSpecification_1173199825608}
{eventFormatterClass com.mycompany.myeventformatterclass}
{maxLogs 0}
{emitterRef AuditServiceProvider_1184686638218}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditEmitters()
```

- Using Jython list:

```
AdminTask.listAuditEmitters()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditEmitters('-interactive')
```

modifyAuditEmitter

The modifyAuditEmitter command modifies the attributes of an audit service provider implementation object.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a audit service provider implementation. (String, required)

Optional parameters

-className
Specifies the class name to use to identify the implementation. (String, optional)

-eventFormatterClass
Specifies the class that implements how the system formats the audit event for output. If you want to use the default audit service provider, do not specify this parameter. (String, optional)

-customProperties
Specifies a list of custom properties formatted as name and value pairs in the following format: `name=value,name=value`. (String, optional)

-auditFilters
Specifies a reference or a group of references to predefined audit filters. Use the following format to specify multiple references: `reference,reference,reference` (String, optional)

-fileLocation
Specifies the location where the system writes the audit logs. (String, optional)

-maxFileSize
Specifies the maximum size that each log reaches before the system saves the audit log with a timestamp. Specify the size in megabytes. The default value is 10 MB. (Integer, optional)

-maxLogs
Specifies the maximum number of log files to create before the system rewrites the oldest audit log. The default value is 100 logs. (Integer, optional)

-wrapBehavior
Specifies a string representing the customizable behavior for binary audit log wrapping. (String, optional).

There are three values for this parameter: WRAP, NOWRAP and SILENT_FAIL

If you use the WRAP option, when the maximum logs are reached, the oldest audit log is rewritten; notification is not sent to the auditor.

The NOWRAP option does not rewrite over the oldest audit log. It stops the audit service, sends a notification to the SystemOut.log, and quiesces the application server.

The SILENT_FAIL option does not rewrite over the oldest audit log. It also stops the audit service, but does allow the WebSphere process to continue. Notifications are not posted in the SystemOut.log.

Return value

The command returns a value of `true` if the system successfully modifies the audit service provider of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditEmitter('-emitterRef AuditServiceProvider_1184686638218
–wrapBehavior NOWRAP -auditFilters AuditSpecification_1173199825608
-fileLocation c:\wasinstall\appserver\profiles\AppSrv01\mylogs -maxFileSize
14 -maxLogs 200')
```

- Using Jython list:

```
AdminTask.modifyAuditEmitter(['-emitterRef', 'AuditServiceProvider_1184686638218',
'–wrapBehavior', 'NOWRAP' '-auditFilters', 'AuditSpecification_1173199825608', '-fileLocation',
'c:\wasinstall\appserver\profiles\AppSrv01\mylogs', '-maxFileSize', '14', '-maxLogs',
'200'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditEmitter('-interactive')
```

setAuditEmitterFilters

The setAuditEmitterFilters command sets the filters for an audit service provider implementation.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies a reference to a audit service provider implementation. (String, required)

-filtersRef
Specifies one or more references to defined audit filters. Use the following format to specify more than one filter reference: `reference,reference,reference` (String, required)

Return value

The command returns a value of `true` if the system successfully sets the filters for the audit service provider.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAuditEmitterFilters('-emitterRef AuditServiceProvider_1173199825608
-filtersRef AuditSpecification_1184598886859')
```

- Using Jython list:

```
AdminTask.setAuditEmitterFilters(['-emitterRef', 'AuditServiceProvider_1173199825608',
'-filtersRef', 'AuditSpecification_1184598886859'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setAuditEmitterFilters('-interactive')
```

AuditSigningCommands command group for the AdminTask object

You can use the Jython scripting language to configure the signing of audit records with the wsadmin tool. Use the commands and parameters in the AuditSigningCommands group to enable, disable, and configure the security audit system to sign audit records.

Use the following commands to configure audit signing:

createAuditSigningConfig

The createAuditSigningConfig command creates the signing model that the system uses to sign the audit records. Use this command to configure your audit signing configuration for the first time. If you have already configured audit signing, use the enableAuditSigning and disableAuditSigning commands to turn audit signing on and off.

You can import the certificate from an existing key file name containing that certificate, automatically generate the certificate, or use the same certificate as the application server uses encrypt the audit records. To use an existing certificate in an existing keystore, specify input values for the -enableAuditEncryption, -certAlias, and –signingKeyStoreRef parameters. Also, set the value of the -useEncryptionCert, –autogenCert, and –importCert parameters as `false` for this scenario.

The user must have the administrator and auditor administrative roles to run this command.

Target object

None.

Required parameters

-enableAuditSigning
Specifies whether to sign audit records. This parameter modifies your audit policy configuration. (Boolean, required)

-certAlias
Specifies the alias name that identifies the generated or imported certificate. (String, required)

-signingKeyStoreRef
Specifies the reference ID of the key store that system imports the certificate to. The signing keystore must already exist in the security.xml file. The system updates this keystore with the certificate that is used to sign the audit records. (String, required)

Optional parameters

-useEncryptionCert
Specifies whether to use the same certificate for encryption and signing. (Boolean, optional)

-autogenCert
Specifies whether to automatically generate the certificate used to sign the audit records. (Boolean, optional)

-importCert
Specifies whether to import an existing certificate to sign the audit records. (Boolean, optional)

-certKeyFileName
Specifies the unique name of the key file for the certificate to import. (String, optional)

-certKeyFilePath
Specifies the key file location for the certificate to import. (String, optional)

-certKeyFileType
Specifies the key file type for the certificate to import. (String, optional)

-certKeyFilePassword
Specifies the key file password for the certificate to import. (String, optional)

-certAliasToImport
Specifies the alias of the certificate to import. (String, optional)

Return value

If successful, returns the shortened from of the keystore where the signing certificate has been added to. Remember, this keystore is in the security.xml file, not the audit.xml file.

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditSigningConfig('-enableAuditSigning true -certAlias
auditSigningCert -autogenCert true -signingKeyStoreRef Ref_Id_of_KeyStoreInSecurityXML')
```

- Using Jython list:

```
AdminTask.createAuditSigningConfig(['-enableAuditSigning', 'true', '-certAlias',
'auditSigningCert', '-autogenCert', 'true -signingKeyStoreRef',
'Ref_Id_of_KeyStoreInSecurityXML'])
```

Interactive mode example usage

- Using Jython :

```
AdminTask.createAuditSigningConfig('-interactive')
```

deleteAuditSigningConfig

The deleteAuditSigningConfig command deletes the signing model that the system uses to sign the audit records. When the system deletes the audit signing configuration, it does not delete the key store file in the security.xml or the signer certificate for the keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully removes the audit signing configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditSigningConfig()
```

- Using Jython list:

```
AdminTask.deleteAuditSigningConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditSigningConfig('-interactive')
```

disableAuditSigning

The disableAuditSigning command disables audit record signing for the security auditing system.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully disables audit signing.

Batch mode example usage

- Using Jython string:

```
AdminTask.disableAuditSigning()
```

- Using Jython list:

```
AdminTask.disableAuditSigning()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.disableAuditSigning('-interactive')
```

enableAuditSigning

The enableAuditSigning command enables audit record signing in the security auditing system.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully enables audit signing in the security auditing system.

Batch mode example usage

- Using Jython string:

```
AdminTask.enableAuditSigning()
```

- Using Jython list:

```
AdminTask.enableAuditSigning()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.enableAuditSigning()
```

getAuditSigningConfig

The getAuditSigningConfig command retrieves the signing model that the system uses to sign the audit records.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes that are associated with the signing model, as the following sample output displays:

```
{{securityXmlSignerScopeName (cell):Node04Cell:(node):Node04}
{securityXmlSignerCertAlias mysigningcert}
{securityXmlSignerKeyStoreName NodeDefaultRootStore}
{signerKeyStoreRef KeyStore_Node04_4}
{enabled true}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditSigningConfig()
```

- Using Jython list:

```
AdminTask.getAuditSigningConfig()
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.getAuditSigningConfig('-interactive')
```

importEncryptionCertificate

The importEncryptionCertificate command imports the self-signed certificate used for encrypting audit data from the encryption keystore into another keystore. Use this command internally to automatically generate a certificate for either encryption or signing. You can also use this command to import the certificate into the keystore by specifying the keyStoreName and keyStoreScope parameters.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name to identify the keystore. (String, required)

-keyFilePath
Specifies the keystore path name that contains the certificate to import. (String, required)

-keyFilePassword
Specifies the password of the keystore that contains the certificate to import. (String, required)

-keyFileType
Specifies the type of the keystore. (String, required)

-certificateAliasFromKeyFile
Specifies the alias of the certificate to import from the keystore file. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-certificateAlias
Specifies a unique name to identify the imported certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully imports the encryption certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.importEncryptionCertificate('{-keyStoreName AuditDefaultKeyStore -keyStoreScope
(cell):Node04Cell -keyFilePath c:/install_root/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12
-keyFilePassword WebAS -keyFileType PKCS12 -certificateAliasFromKeyFile root -certificateAlias myimportcert')
```

- Using Jython list:

```
AdminTask.importEncryptionCertificate(['{-keyStoreName', 'AuditDefaultKeyStore', '-keyStoreScope',
'(cell):Node04Cell', '-keyFilePath', 'c:/install_root/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12',
'-keyFilePassword', 'WebAS', '-keyFileType', 'PKCS12', '-certificateAliasFromKeyFile',
'root', '-certificateAlias', 'myimportcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importEncryptionCertificate('-interactive')
```

isAuditSigningEnabled

The isAuditSigningEnabled command indicates whether audit signing is enabled or disable in the security audit system.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if signing is configured in the security auditing system.

Batch mode example usage

- Using Jython string:

```
AdminTask.isAuditSigningEnabled()
```

- Using Jython list:

```
AdminTask.isAuditSigningEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isAuditSigningEnabled('-interactive')
```

modifyAuditSigningConfig

The modifyAuditSigningConfig command modifies the signing model that the system uses to sign the audit records.

The certificate may either be imported from an existing key file name containing that certificate, automatically generated, or be the same certificate used to encrypt the audit records. To use an existing certificate in an existing keystore, specify input values for the -enableAuditEncryption, -certAlias, and –signingKeyStoreRef parameters. Also, set the value the -useEncryptionCert, –autogenCert, and –importCert parameters as `false` for this scenario.

The user must have the administrator and auditor administrative roles to run this command.

Target object

None.

Required parameters

-enableAuditSigning
Specifies whether to sign audit records. This parameter modifies your audit policy configuration. (Boolean, required)

-certAlias
Specifies the alias name that identifies the generated or imported certificate. (String, required)

-signingKeyStoreRef
Specifies the reference ID of the key store that system imports the certificate to. The signing keystore must already exist in the security.xml file. The system updates this keystore with the certificate that is used to sign the audit records. (String, required)

Optional parameters

-useEncryptionCert
Specifies whether to use the same certificate for encryption and signing. (Boolean, optional)

-autogenCert
Specifies whether to automatically generate the certificate used to sign the audit records. (Boolean, optional)

-importCert
Specifies whether to import an existing certificate to sign the audit records. (Boolean, optional)

-certKeyFileName
Specifies the unique name of the key file for the certificate to import. (String, optional)

-certKeyFilePath
Specifies the key file location for the certificate to import. (String, optional)

-certKeyFileType
Specifies the key file type for the certificate to import. (String, optional)

-certKeyFilePassword
Specifies the key file password for the certificate to import. (String, optional)

-certAliasToImport
Specifies the alias of the certificate to import. (String, optional)

Return value

The command returns a value of `true` if the system successfully modifies the security auditing system configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditSigningConfig('-enableAuditSigning true -certAlias auditSigningCert
-autogenCert true -signingKeyStoreRef Ref_Id_of_KeyStoreInSecurityXML')
```

- Using Jython list:

```
AdminTask.modifyAuditSigningConfig(['-enableAuditSigning', 'true', '-certAlias',
'auditSigningCert', '-autogenCert', 'true', '-signingKeyStoreRef', 'Ref_Id_of_KeyStoreInSecurityXML'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditSigningConfig('-interactive')
```

AuditEncryptionCommands command group for the AdminTask object

You can use the Jython scripting language to configure the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditEncryptionCommands group to configure the security audit system to encrypt audit records.

Use the following commands to enable, disable, and configure audit record encryption:

createAuditEncryptionConfig

The createAuditEncryptionConfig command creates the encryption model used to encrypt the audit records.

You can import the certificate from an existing key file name containing that certificate or automatically generate a certificate.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-enableAuditEncryption
Specifies whether to encrypt audit records. This parameter modifies your audit policy configuration. (Boolean, required)

-certAlias
Specifies the alias name that identifies the generated or imported certificate. (String, required)

-encryptionKeyStoreRef
Specifies the reference ID of the keystore to import the certificate to. (String, required)

Optional parameters

-autogenCert
Specifies whether to automatically generate the certificate used to encrypt the audit records. You must specify either this parameter or the -importCert parameter, but you cannot specify both. (Boolean, optional)

-importCert
Specifies whether to import an existing certificate to encrypt the audit records. You must specify either this parameter or the -autogenCert parameter, but you cannot specify both. (Boolean, optional)

-certKeyFileName
Specifies the unique name of the key file for the certificate to import. (String, optional)

-certKeyFilePath
Specifies the key file location for the certificate to import. (String, optional)

-certKeyFileType
Specifies the key file type for the certificate to import. (String, optional)

-certKeyFilePassword
Specifies the key file password for the certificate to import. (String, optional)

-certAliasToImport
Specifies the alias of the certificate to import. (String, optional)

Return value

The command returns the shortened form of the reference ID of the created encryption keystore if the system successfully creates the audit encryption configuration, as the following example output displays:

```
KeyStore_1173199825578
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditEncryptionConfig('-enableAuditEncryption true -certAlias
auditCertificate -autogenCert true -encryptionKeyStoreRef auditKeyStore')
```

- Using Jython list:

```
AdminTask.createAuditEncryptionConfig(['-enableAuditEncryption', 'true', '-certAlias',
'auditCertificate', '-autogenCert', 'true', '-encryptionKeyStoreRef', 'auditKeyStore'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.createAuditEncryptionConfig('-interactive')
```

createAuditSelfSignedCertificate

The createAuditSelfSignedCertificate command creates a self-signed certificate. Use this command internally to automatically generate a certificate for encryption and signing or to import that certificate into the keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore where the system imports the self-signed certificate to. (String, optional)

-certificateAlias
Specifies a unique alias name for the certificate. (String, required)

-certificateSize
Specifies the size that the private key uses for the personal certificate. The default value is `1024`. (Integer, required)

-certificateCommonName
Specifies the common name portion of the distinguished name. (String, required)

Optional parameters

-certificateOrganization
Specifies the organizational part of the distinguished name. (String, optional)

-keyStoreScope
Specifies the scope of the keystore that the system imports the self-signed certificate to. (String, optional)

-certificateVersion
Specifies the version of the personal certificate. (String, optional)

-certificateOrganizationalUnit
Specifies the organization unit part of the distinguished name. (String, optional)

-certificateLocality
Specifies the locality portion of the distinguished name. (String, optional)

-certificateState
Specifies the state portion of the distinguished name. (String, optional)

-certificateZip
Specifies the zip code portion of the distinguished name. (String, optional)

-certificateCountry
Specifies the country portion of the distinguished name. The default value is `US`. (String, optional)

-certificateValidDays
Specifies the length of time, in days, which the certificate is valid. The default value is `365` days. (Integer, optional)

Return value

The command returns a value of `true` if the system successfully creates the self-signed certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditSelfSignedCertificate('-keyStoreName AuditDefaultKeyStore -keyStoreScope
(cell):Node04Cell -certificateAlias myNew -certificateCommonName cn=oet -certificateOrganization mycompany')
```

- Using Jython list:

```
AdminTask.createAuditSelfSignedCertificate(['-keyStoreName', 'AuditDefaultKeyStore', '-keyStoreScope',
'(cell):Node04Cell', '-certificateAlias', 'myNew', '-certificateCommonName', 'cn=oet',
'-certificateOrganization', 'mycompany'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditSelfSignedCertificate('-interactive')
```

deleteAuditCertificate

The deleteAuditCertificate command deletes a self-signed certificate from an audit keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore from which the system deletes the self-signed certificate. (String, required)

-certificateAlias
Specifies a unique alias name for the certificate to delete. (String, required)

Optional parameters

-keyStoreScope
Specifies a unique alias name for the certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully deletes the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditCertificate('-keyStoreName myKeystore -certificateAlias oldCertificate')
```

- Using Jython list:

```
AdminTask.deleteAuditCertificate(['-keyStoreName', 'myKeystore', '-certificateAlias', 'oldCertificate'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditCertificate('-interactive')
```

deleteAuditEncryptionConfig

The deleteAuditEncryptionConfig command deletes the encryption model used to encrypt the audit records. The command does not remove keystore files or the certificates.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully deletes the audit encryption configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditEncryptionConfig()
```

- Using Jython list:

```
AdminTask.deleteAuditEncryptionConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditEncryptionConfig('-interactive')
```

disableAuditEncryption

The disableAuditEncryption command disables the encryption of audit records.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully disables audit record encryption.

Batch mode example usage

- Using Jython string:

```
AdminTask.disableAuditEncryption()
```

- Using Jython list:

```
AdminTask.disableAuditEncryption()
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.disableAuditEncryption('-interactive')
```

enableAuditEncryption

The enableAuditEncryption command enables the encryption of audit records.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully enables audit record encyption.

Batch mode example usage

- Using Jython string:

```
AdminTask.enableAuditEncryption()
```

- Using Jython list:

```
AdminTask.enableAuditEncryption()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.enableAuditEncryption()
```

exportAuditCertificate

The exportAuditCertificate command exports a self-signed certificate from a keystore. To use this command, you must adhere to the following user role and privilege guidelines:

- You must have audit privileges to export the certificate from an audit keystore.
- You must have the auditor and administrator roles to export the certificate to a security keystore.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

-keyStorePassword
Specifies the password that the system uses to access the keystore specified with the -keyStoreName parameter. (String, required)

-keyFilePath
Specifies the key store path name that contains the certificate to export. (String, required)

-keyFilePassword
Specifies the password of the keystore that contains the certificate to export. (String, required)

-keyFileType
Specifies the type of the keystore. (String, required)

-certificateAlias
Specifies the alias of the certificate to export from the keystore. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-aliasInKeyStore
Specifies a new unique name to identify the exported certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully exports the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportAuditCertificate('-keyStoreName AuditDefaultKeyStore -keyStoreScope
(cell):Node04Cell -keyFilePath c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12
-keyFilePassword myPwd -keyFileType PKCS12 -certificateAlias root')
```

- Using Jython list:

```
AdminTask.exportAuditCertificate(['-keyStoreName', 'AuditDefaultKeyStore', '-keyStoreScope',
'(cell):Node04Cell', '-keyFilePath', 'c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12',
'-keyFilePassword', 'myPwd', '-keyFileType', 'PKCS12', '-certificateAlias', 'root'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportAuditCertificate('-interactive')
```

exportAuditCertToManagedKS

The exportAuditCertToManagedKS command exports a self-signed certificate from an audit keystore to a managed audit keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the managed keystore. (String, required)

-keyStorePassword
Specifies the password of the managed keystore that contains the certificate to export. (String, required)

-toKeyStoreName
Specifies the unique name of the managed keystore that contains the certificate to export. (String, required)

-certificateAlias
Specifies a unique name to identify the exported certificate. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-toKeyStoreScope
Specifies the scope of the managed keystore that contains the certificate to export. (String, optional)

-aliasInKeyStore
Specifies the new unique name to identify the exported certificate. If you do not specify a value for this parameter, the system sets the unique name to the value specified for the -certificateAlias parameter. (String, optional)

Return value

The command returns a value of `true` if the system successfully exports the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportAuditCertToManagedKS('-keyStoreName  auditEncryptionKeyStore -keyStorePassword myPwd
-toKeyStoreName  AuditTrustStore -toKeyStoreScope (cell):my03Cell -certificateAlias newauditcert
-aliasInKeyStore newauditcert1')
```

- Using Jython list:

```
AdminTask.exportAuditCertToManagedKS(['-keyStoreName', 'auditEncryptionKeyStore', '-keyStorePassword', 'myPwd',
'-toKeyStoreName', 'AuditTrustStore', '-toKeyStoreScope', '(cell):my03Cell', '-certificateAlias', 'newauditcert',
'-aliasInKeyStore', 'newauditcert1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportAuditCertToManagedKS('-interactive')
```

getAuditCertificate

The getAuditCertificate command retrieves the attributes for an audit self-signed certificate in an audit keystore.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the managed keystore of interest. (String, required)

-certificateAlias
Specifies a unique name to identify the exported certificate of interest. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore of interest. (String, optional)

Return value

The command returns a list of attributes associated with the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditCertificate('-keyStoreName auditEncryptionKeyStore -certificateAlias
newauditcert')
```

- Using Jython list:

```
AdminTask.getAuditCertificate(['-keyStoreName', 'auditEncryptionKeyStore', '-certificateAlias',
'newauditcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditCertificate('-interactive')
```

getAuditEncryptionConfig

The getAuditEncryptionConfig command retrieves the encryption model that the system uses to encrypt the audit records.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes associated with the encryption model, as the following example output displays:

```
{{certRef Certificate_1184698729015}
{keystoreRef KeyStore_1173199825578}
{keyStore AuditDefaultKeyStore(cells/CHEYENNENode04Cell|audit.xml#KeyStore_1173199825578)}
{enabled true}
```

```
{alias mycertalias}
{_Websphere_Config_Data_Version {}}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#Certificate_1184698729015}
{_Websphere_Config_Data_Type Certificate}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEncryptionConfig()
```

- Using Jython list:

```
AdminTask.getAuditEncryptionConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEncryptionConfig('-interactive')
```

getEncryptionKeyStore

The getEncryptionKeyStore command retrieves the attributes for the keystore that contains the certificate that the system uses to encrypt the audit records.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes for the keystore of interest, as the following example displays:

```
{{location ${CONFIG_ROOT}/audittrust.p12}
{password *****}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#KeyStore_1173199825578}
{_Websphere_Config_Data_Version {}}
{useForAcceleration false}
{slot 0}
{type PKCS12}
{additionalKeyStoreAttrs {}}
{fileBased true}
{_Websphere_Config_Data_Type KeyStore}
{customProviderClass {}}
{hostList {}}
{keystoreRef KeyStore_1173199825578}
{createStashFileForCMS false}
{description {keyStore description}}
{managementScope (cells/CHEYENNENode04Cell|audit.xml#ManagementScope_1173199825608)}
{readOnly false}
{initializeAtStartup true}
{usage {}}
{provider IBMJCE}
{name AuditDefaultKeyStore}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getEncryptionKeyStore()
```

- Using Jython list:

```
AdminTask.getEncryptionKeyStore()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEncryptionKeyStore('-interactive')
```

importAuditCertFromManagedKS

The importAuditCertFromManagedKS command imports a self-signed certificate into a keystore from a managed audit keystore. Use this command internally to automatically generate a certificate for encryption or signing and to import a certificate into the keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the managed keystore. (String, required)

-fromKeyStoreName
Specifies the unique name of the managed keystore that contains the certificate to import. (String, required)

-fromKeyStorePassword
Specifies the password of the managed keystore that contains the certificate to import. (String, required)

-certificateAliasFromKeyFile
Specifies the alias of the certificate to import from the managed keystore file. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-fromKeyStoreScope
Specifies the scope of the managed keystore that contains the certificate to import. (String, optional)

-certificateAlias
Specifies a unique name to identify the imported certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully imports the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.importAuditCertFromManagedKS('-keyStoreName AuditDefaultKeyStore -keyStoreScope
(cell):myNode03Cell -fromKeyStoreName AuditSecondDefaultKeyStore -fromKeyStoreScope
(cell):myNode03Cell -fromKeyStorePassword myPwd
-certificateAliasFromKeyFile root -certificateAlias myimportcert')
```

- Using Jython list:

```
AdminTask.importAuditCertFromManagedKS(['-keyStoreName', 'AuditDefaultKeyStore', '-keyStoreScope',
'(cell):Node04Cell', '-fromKeyStoreName', 'AuditSecondDefaultKeyStore', '-fromKeyStoreScope',
'(cell):myNode03Cell', '-fromKeyStorePassword', 'myPwd', '-certificateAliasFromKeyFile',
'root', '-certificateAlias', 'myimportcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importAuditCertFromManagedKS('-interactive')
```

importAuditCertificate

The importAuditCertificate command imports a self-signed certificate into a keystore. Use this command internally to automatically generate a certificate for encryption or signing and to import a certificate into the keystore. To use this command, you must adhere to the following user role and privilege guidelines:

- You must have audit privileges to import the certificate to an audit keystore.
- You must have the auditor and administrator roles to import the certificate to a security keystore.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

-keyFilePath
Specifies the key store path name that contains the certificate to import. (String, required)

-keyFilePassword
Specifies the password of the keystore that contains the certificate to import. (String, required)

-keyFileType
Specifies the type of the keystore. (String, required)

-certificateAliasFromKeyFile
Specifies the alias of the certificate to import from the keystore file. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-certificateAlias
Specifies a unique name to identify the imported certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully imports the audit certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.importAuditCertificate('-keyStoreName AuditDefaultKeyStore -keyStoreScope
(cell):Node04Cell -keyFilePath c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12
-keyFilePassword myPwd -keyFileType PKCS12 -certificateAliasFromKeyFile root -certificateAlias myimportcert')
```

- Using Jython list:

```
AdminTask.importAuditCertificate(['-keyStoreName', 'AuditDefaultKeyStore', '-keyStoreScope', '(cell):Node04Cell',
'-keyFilePath', 'c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12',
'-keyFilePassword', 'myPwd', '-keyFileType', 'PKCS12', '-certificateAliasFromKeyFile', 'root',
'-certificateAlias', 'myimportcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importAuditCertificate('-interactive')
```

importEncryptionCertificate

The importEncryptionCertificate command imports the self-signed certificate that the system uses to encrypt audit data from the encryption keystore into a managed keystore in security.xml.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

-keyFilePath
Specifies the key store path name that contains the certificate to import. (String, required)

-keyFilePassword
Specifies the password of the keystore that contains the certificate to import. (String, required)

-keyFileType
Specifies the type of the keystore. (String, required)

-certificateAliasFromKeyFile
Specifies the alias of the certificate to import from the keystore file. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore. (String, optional)

-certificateAlias
Specifies a unique name to identify the imported certificate. (String, optional)

Return value

The command returns a value of `true` if the system successfully imports the encryption certificate.

Batch mode example usage

- Using Jython string:

```
AdminTask.importEncryptionCertificate('-keyStoreName DefaultKeyStore -keyStoreScope (cell):Node04Cell
-keyFilePath c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12
-keyFilePassword myPwd -keyFileType PKCS12 -certificateAliasFromKeyFile root -certificateAlias myimportcert')
```

- Using Jython list:

```
AdminTask.importEncryptionCertificate(['-keyStoreName', 'DefaultKeyStore', '-keyStoreScope', '(cell):Node04Cell',
'-keyFilePath', 'c:/wasinstall/appserver/profiles/AppSrv01/config/cells/Node04Cell/nodes/Node04/trust.p12',
'-keyFilePassword', 'myPwd', '-keyFileType', 'PKCS12', '-certificateAliasFromKeyFile', 'root',
'-certificateAlias', 'myimportcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importEncryptionCertificate('-interactive')
```

isAuditEncryptionEnabled

The isAuditEncryptionEnabled command determines if audit record encryption is enabled.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if audit record encryption is enabled.

Batch mode example usage

- Using Jython string:

```
AdminTask.isAuditEncryptionEnabled()
```

- Using Jython list:

```
AdminTask.isAuditEncryptionEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isAuditEncryptionEnabled('-interactive')
```

listAuditEncryptionKeyStores

The listAuditEncryptionKeyStores command retrieves the attributes for each configured encryption keystore from the `audit.xml` file. The command returns attributes for active and inactive keystores.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes for each configured keystore, as the following example output displays:

```
{{location ${CONFIG_ROOT}/audittrust.p12}
{password *****}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#KeyStore_1173199825578}
{useForAcceleration false}
{slot 0}
{type PKCS12}
{additionalKeyStoreAttrs {}}
{fileBased true}
{_Websphere_Config_Data_Type KeyStore}
{customProviderClass {}}
{hostList {}}
{keystoreRef KeyStore_1173199825578}
{createStashFileForCMS false}
{description {keyStore description}}
{readOnly false}
{initializeAtStartup true}
{managementScope (cells/CHEYENNENode04Cell|audit.xml#ManagementScope_1173199825608)}
{usage {}}
{provider IBMJCE}
{name AuditDefaultKeyStore}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditEncryptionKeyStores()
```

- Using Jython list:

```
AdminTask.listAuditEncryptionKeyStores()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditEncryptionKeyStores('-interactive')
```

listCertAliases

The listCertAliases command retrieves a list of the personal certificates in the keystore, as specified by the keystore name and scope of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-keyStoreName
Specifies the unique name of the keystore. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope of the keystore. The default value is the cell scope. (String, optional)

Return value

The command returns a list of certificate aliases for the personal certificates that are configured for the keystore, as the following sample output displays:

```
mycertalias
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listCertAliases('-keyStoreName AuditDefaultKeyStore -keyStoreScope (cell):Node04Cell')
```

- Using Jython list:

```
AdminTask.listCertAliases(['-keyStoreName AuditDefaultKeyStore -keyStoreScope (cell):Node04Cell'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listCertAliases('-interactive')
```

modifyAuditEncryptionConfig

The modifyAuditEncryptionConfig command modifies the encryption model that the system uses to encrypt the audit records. Specify values for the -enableAuditEncryption, -certAlias, and encryptionKeyStoreRef parameters to use an existing keystore. Do not specify the -importCert or -autogenCert parameters if you use an existing keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

None

Optional parameters

-enableAuditEncryption
Specifies whether to encrypt audit records. This parameter modifies your audit policy configuration. (Boolean, optional)

-autogenCert
Specifies whether to automatically generate the certificate used to encrypt the audit records. You must specify either this parameter or the -importCert parameter, but you cannot specify both. (Boolean, optional)

-importCert
Specifies whether to import an existing certificate to encrypt the audit records. You must specify either this parameter or the -autogenCert parameter, but you cannot specify both. (Boolean, optional)

-certKeyFileName
Specifies the unique name of the key file for the certificate to import. (String, optional)

-certKeyFilePath
Specifies the key file location for the certificate to import. (String, optional)

-certKeyFileType
Specifies the key file type for the certificate to import. (String, optional)

-certKeyFilePassword
Specifies the key file password for the certificate to import. (String, optional)

-certAliasToImport
Specifies the alias of the certificate to import. (String, optional)

-certAlias
Specifies the alias name that identifies the generated or imported certificate. (String, optional)

-encryptionKeyStoreRef
Specifies the reference ID of the keystore to import the certificate to. (String, optional)

Return value

The command returns a value of `true` if the system successfully updates the configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditEncryptionConfig('-enableAuditEncryption true -certAlias mycertalias
-encryptionKeyStoreRef KeyStore_1173199825578')
```

- Using Jython list:

```
AdminTask.modifyAuditEncryptionConfig(['-enableAuditEncryption', 'true', '-certAlias', 'mycertalias',
'-encryptionKeyStoreRef', 'KeyStore_1173199825578'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditEncryptionConfig('-interactive')
```

renewAuditCertificate

The renewAuditCertificate command renews a self signed certificate in an audit keystore.

The user must have the auditor administrative role to run this command.

Target object

None.

-keyStoreName
Specifies the unique name of the managed keystore of interest. (String, required)

-certificateAlias
Specifies a unique name to identify the exported certificate to renew. (String, required)

Optional parameters

-keyStoreScope
Specifies the scope name of the keystore of interest. (String, optional)

Return value

The command returns a value of `true` if the system successfully updates the configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.renewAuditCertificate('-keyStoreName auditEncryptionKeyStore
-certificateAlias newauditcert')
```

- Using Jython list:

```
AdminTask.renewAuditCertificate(['-keyStoreName', 'auditEncryptionKeyStore',
'-certificateAlias', 'newauditcert'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.renewAuditCertificate('-interactive')
```

AuditEventFactoryCommands for the AdminTask object

You can use the Jython scripting language to configure the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditEventFactoryCommands group to configure the default or a third-party audit event factory.

Use the following commands to configure the default audit event factory or a third-party audit event factory:

- "createAuditEventFactory"
- "deleteAuditEventFactoryByName" on page 333
- "deleteAuditEventFactoryByRef" on page 334
- "getAuditEventFactory" on page 334
- "getAuditEventFactoryClass" on page 335
- "getAuditEventFactoryFilters" on page 336
- "getAuditEventFactoryName" on page 336
- "getAuditEventFactoryProvider" on page 337
- "listAuditEventFactories" on page 337
- "modifyAuditEventFactory" on page 338
- "setAuditEventFactoryFilters" on page 339

createAuditEventFactory

The createAuditEventFactory command creates an audit event factory in your security auditing system configuration. You can use the default implementation of the audit event factory or use a third-party implementation. To configure a third-party implementation, use the optional -customProperties parameter to specify any properties necessary to configure the audit event factory implementation.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies a unique name that identifies the audit event factory. (String, required)

-className
Specifies the class implementation of the audit event factory interface. (String, required)

-provider
Specifies a reference to a predefined audit service provider implementation. (String, required)

-auditFilters
Specifies a reference or a group of references to predefined audit filters, using the following format: `reference, reference, reference` (String, required)

Optional parameters

-customProperties
Specifies any custom properties necessary to configure a third-party implementation. (String, optional)

Return value

The command returns the shortened reference ID for the newly created audit event factory.

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditEventFactory('-uniqueName myeventfactory -className
com.mycompany.myeventfactoryclass -provider AuditServiceProvider_1173199825608
-customProperties a=b -auditFilters AuditSpecification_1184598886859')
```

- Using Jython list:

```
AdminTask.createAuditEventFactory(['-uniqueName', 'myeventfactory', '-className',
'com.mycompany.myeventfactoryclass', '-provider', 'AuditServiceProvider_1173199825608',
'-customProperties', 'a=b', '-auditFilters', 'AuditSpecification_1184598886859'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditEventFactory()
```

deleteAuditEventFactoryByName

The deleteAuditEventFactoryByName command deletes the audit event factory implementation in the `audit.xml` file that matches a specific unique name identifier.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-uniqueName
Specifies the unique name of the audit event factory implementation. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit event factory.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditEventFactoryByName('–uniqueName
myeventfactory')
```

- Using Jython list:

```
AdminTask.deleteAuditEventFactoryByName(['–uniqueName', 'myeventfactory'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditEventFactoryByName('-interactive')
```

deleteAuditEventFactoryByRef

The deleteAuditEventFactoryByRef command deletes the audit event factory implementation that matches the reference ID of interest.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit event factory.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditEventFactoryByRef('–eventFactoryRef
AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.deleteAuditEventFactoryByRef(['–eventFactoryRef', 'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditEventFactoryByRef('–interactive')
```

getAuditEventFactory

The getAuditEventFactory command retrieves the list of attributes for the audit event factory implementation in the `audit.xml` file for a specific reference id.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns an attribute list for the audit event factory implementation of interest, as the following example output displays:

```
{{name myeventfactory}
{properties {{{validationExpression {}}
{name a}
{description {}}
{value b}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#Property_1184688593531}
{_Websphere_Config_Data_Type Property}
{required false}}}}
{className com.mycompany.myeventfactoryclass}
{auditServiceProvider auditServiceProviderImpl_1(cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608)}
{auditSpecifications DefaultAuditSpecification_1(cells/Node04Cell|audit.xml#AuditSpecification_1173199825608)}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditEventFactory_1184688293515}
{_Websphere_Config_Data_Type AuditEventFactory}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEventFactory('-eventFactoryRef AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.getAuditEventFactory(['-eventFactoryRef', 'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEventFactory('-interactive')
```

getAuditEventFactoryClass

The getAuditEventFactoryClass command retrieves the class name of the audit event factory implementation that matches a specific reference ID in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns the class name of the audit event factory of interest, as the following sample output displays:

```
com.mycompany.myeventfactoryclass
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEventFactoryClass('-eventFactoryRef
AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.getAuditEventFactoryClass(['-eventFactoryRef',
'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEventFactoryClass('-interactive')
```

getAuditEventFactoryFilters

The getAuditEventFactoryFilters command retrieves a list of defined filters for the passed-in event factory.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns a list of the defined filters for the event factory reference of interest in a shortened format, as the following sample output displays:

```
AUTHN:SUCCESS,AUTHN:INFO,AUTHZ:SUCCESS,AUTHZ:INFO
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEventFactoryFilters('-eventFactoryRef
AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.getAuditEventFactoryFilters(['-eventFactoryRef',
'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEventFactoryFilters('-interactive')
```

getAuditEventFactoryName

The getAuditEventFactoryName command retrieves the unique name of the audit event factory implementation that matches a specific reference ID in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns the name of the audit event factory, as the following sample output displays:

```
myeventfactory
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEventFactoryName('-eventFactoryRef
AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.getAuditEventFactoryName(['-eventFactoryRef',
'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEventFactoryName('-interactive')
```

getAuditEventFactoryProvider

The getAuditEventFactoryProvider command retrieves the object name of the audit service provider that a specific audit event factory implementation uses in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Return value

The command returns the object name of the audit service provider for the audit event factory of interest, as the following sample output displays:

```
auditServiceProviderImpl_1(cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608)
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditEventFactoryProvider('-eventFactoryRef
AuditEventFactory_1184688293515')
```

- Using Jython list:

```
AdminTask.getAuditEventFactoryProvider(['-eventFactoryRef',
'AuditEventFactory_1184688293515'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditEventFactoryProvider('-interactive')
```

listAuditEventFactories

The listAuditEventFactories command retrieves a list of audit event factory objects and their attributes that are defined in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns an array list of audit event factories and attributes, as the following example output displays:

```
{{auditSpecifications DefaultAuditSpecification_1(cells/Node04Cell|audit.xml#AuditSpecification_1173199825608)
DefaultAuditSpecification_2(cells/Node04Cell|audit.xml#AuditSpecification_1173199825609)
DefaultAuditSpecification_3(cells/Node04Cell|audit.xml#AuditSpecification_1173199825610)
DefaultAuditSpecification_4(cells/Node04Cell|audit.xml#AuditSpecification_1173199825611)}
{name auditEventFactoryImpl_1}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditEventFactory_1173199825608}
{_Websphere_Config_Data_Type AuditEventFactory}
{auditSpecRef4 AuditSpecification_1173199825611}
{properties {}}
{auditSpecRef3 AuditSpecification_1173199825610}
{className com.ibm.ws.security.audit.AuditEventFactoryImpl}
{auditServiceProvider auditServiceProviderImpl_1(cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608)}
{auditSpecRef2 AuditSpecification_1173199825609}
{auditSpecRef1 AuditSpecification_1173199825608}
{auditEventFactoryRef AuditEventFactory_1173199825608}
{emitterRef AuditServiceProvider_1173199825608}}
{{auditSpecifications myfilter(cells/Node04Cell|audit.xml#AuditSpecification_1184598886859)}
{name myeventfactory}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditEventFactory_1184688293515}
{_Websphere_Config_Data_Type AuditEventFactory}
{className com.mycompany.myeventfactoryclass}
{auditServiceProvider auditServiceProviderImpl_1(cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608)}
{properties {{{validationExpression {}}
{name a}
{description {}}
{value b}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#Property_1184688293546}
{_Websphere_Config_Data_Type Property}
{required false}}}}
{auditSpecRef1 AuditSpecification_1184598886859}
{auditEventFactoryRef AuditEventFactory_1184688293515}
{emitterRef AuditServiceProvider_1173199825608}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditEventFactories()
```

- Using Jython list:

```
AdminTask.listAuditEventFactories()
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.listAuditEventFactories('-interactive')
```

modifyAuditEventFactory

The modifyAuditEventFactory command modifies the attributes of the audit event factory implementation that the command references with the reference id.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

Optional parameters

-provider
Specifies the reference ID of an audit service provider implementation. (String, optional)

-className
Specifies the name of the class that implements the audit event factory interface. (String, optional)

-customProperties
Specifies one or more custom properties to associate with the audit event factory of interest. Use the following format: `name=value, name=value` (String, optional)

-auditFilters
Specifies a list of references to audit filters that exist in your configuration. You can separate each item in the list with a comma (`,`), a semicolon (`;`), or a space. (String, optional)

Return value

The command returns a value of `true` if the system successfully updates the security auditing system configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditEventFactory('-eventFactoryRef AuditEventFactory_1184688293515 -provider
AuditServiceProvider_1173199825608 -customProperties b=c')
```

- Using Jython list:

```
AdminTask.modifyAuditEventFactory(['-eventFactoryRef', 'AuditEventFactory_1184688293515',
'-provider', 'AuditServiceProvider_1173199825608', '-customProperties', 'b=c'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditEventFactory('-interactive')
```

setAuditEventFactoryFilters

The setAuditEventFactoryFilters command sets the filters for an audit event factory implementation.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-eventFactoryRef
Specifies an audit event factory implementation. This parameter can be a reference to the event factory object. (String, required)

-filtersRef
Specifies a list of references to defined audit filters. (String, required)

Return value

The command returns a value of `true` if the system successfully sets the filters for the audit event factory.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAuditEventFactoryFilters('-eventFactoryRef AuditEventFactory_1184688293515
-filtersRef AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.setAuditEventFactoryFilters(['-eventFactoryRef', 'AuditEventFactory_1184688293515',
'-filtersRef', 'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.setAuditEventFactoryFilters('-interactive')
```

AuditFilterCommands command group for the AdminTask object

You can use the Jython scripting language to configure the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditFilterCommands group to configure and manage auditable events.

Use the following commands to configure filters for auditable events in your security auditing configuration:

convertFilterRefToString

The convertFilterRefToString command converts a reference ID of a filter to a shortened string value such as `AUTHN:SUCCESS`.

Target object

None.

Required parameters

`-filterRef`
Specifies a reference ID for a specific audit filter in the `audit.xml` file. The system defines 4 default audit filters by default. Use the createAuditFilter command to create additional audit filters in your `audit.xml` configuration file. (String, required)

Return value

The command returns the string value of an event type in a shortened format, as the following sample output displays:

```
AUTHN:SUCCESS,AUTHN:INFO,AUTHZ:SUCCESS,AUTHZ:INFO
```

Batch mode example usage

- Using Jython string:

```
AdminTask.convertFilterRefToString('-filterRef AuditSpecification_1184598886859')
```

- Using Jython list:

```
AdminTask.convertFilterRefToString(['-filterRef', 'AuditSpecification_1184598886859'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.convertFilterRefToString('-interactive')
```

convertFilterStringToRef

The convertFilterStringToRef command converts the shortened name of an event type, such as AUTHN:SUCCESS, to the reference ID of the audit filter in the audit.xml configuration file.

The command accepts one event and outcome pair. The command does not accept multiple event and outcome pairs, such as AUTHN:SUCCESS AUTHZ:SUCCESS.

Target object

None.

Required parameters

-filter
Specifies a shortened form of a reference ID for an audit filter, such as AUTHN:SUCCESS. The event type must exist in your security auditing system configuration. (String, required)

Return value

The command returns the reference ID for the event type of interest, as the following example displays:

```
AuditSpecification_1173199825608
```

Batch mode example usage

- Using Jython string:

```
AdminTask.convertFilterStringToRef('-filter AUTHN:SUCCESS')
```

- Using Jython list:

```
AdminTask.convertFilterStringToRef(['-filter', 'AUTHN:SUCCESS'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.convertFilterStringToRef('-interactive')
```

createAuditFilter

The createAuditFilter command creates and enables a new audit event filter specification entry in the audit.xml configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-name
Specifies a unique name to associate with the audit event filter. (String, required)

-eventType
Specifies a list of one or more auditable events. To specify a list, separate each outcome with a comma (**,**) character. (String, required)

You can configure the following auditable events in your security auditing system:

Table 52. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

Important: The following security audit event types are not used in this release of WebSphere Application Server:

- SECURITY_MGMT_KEY
- SECURITY_RUNTIME_KEY
- SECURITY_MGMT_PROVISIONING
- SECURITY_MGMT_REGISTRY
- SECURITY_RUNTIME

-outcome
Specifies a list of one or multiple event outcomes. For each audit event type, you must specify an outcome. Valid outcomes include `SUCCESS`, `FAILURE`, `REDIRECT`, `ERROR`, `DENIED`, `WARNING`, and `INFO`. (String, required)

Return value

If the system is successful, the command returns the reference ID for the new audit event filter, as the following sample output displays:

```
AuditSpecification_1184689433421
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditFilter('-name myfilter -eventType
"SECURITY_MGMT_PROVISIONING, SECURITY_MGMT_POLICY" -outcome SUCCESS')
```

- Using Jython list:

```
AdminTask.createAuditFilter(['-name', 'myfilter', '-eventType',
'"SECURITY_MGMT_PROVISIONING,', 'SECURITY_MGMT_POLICY"', '-outcome', 'SUCCESS'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditFilter('-interactive')
```

deleteAuditFilter

The deleteAuditFilter command deletes the audit event filter specification from the `audit.xml` file that the system references by an event type and outcome.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-eventType
Specifies the auditable event to delete. (String, required)

The following table displays all valid event types:

Table 53. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

-outcome
Specifies the event outcome. Valid outcomes include `SUCCESS`, `FAILURE`, `REDIRECT`, `ERROR`, `DENIED`, `WARNING`, and `INFO`. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit filter from your configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditFilter('-eventType SECURITY_AUTHN –outcome SUCCESS')
```

- Using Jython list:

```
AdminTask.deleteAuditFilter(['-eventType', 'SECURITY_AUTHN', '–outcome', 'SUCCESS'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditFilter('-interactive')
```

deleteAuditFilterByRef

The deleteAuditFilterByRef command deletes the audit filter that the system references by the referenced id.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-filterRef
Specifies the reference ID for an audit filter in your security auditing system configuration. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit filter specification from the `audit.xml` file.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditFilterByRef('-filterRef AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.deleteAuditFilterByRef(['-filterRef', 'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditFilterByRef('-interactive')
```

disableAuditFilter

The disableAuditFilter command disables the audit filter specification that corresponds to a specific reference id.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-filterRef
Specifies a reference ID for a specific audit filter in the `audit.xml` file. The system defines 4 default audit filters by default. Use the createAuditFilter command to create additional audit filters in your `audit.xml` configuration file. (String, required)

Return value

The command returns a value of `true` if the system successfully disables the audit filter.

Batch mode example usage

- Using Jython string:

```
AdminTask.disableAuditFilter('-filterRef', 'AuditSpecification_1184689433421')
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.disableAuditFilter('-interactive')
```

enableAuditFilter

The enableAuditFilter command enables the audit filter specification that corresponds to a specific reference id. Use this command to enable a filter that was previously configured and disabled in your security auditing system configuration. To create a new audit filter specification, use the creatAuditFilter command.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-filterRef
Specifies a reference ID for a specific audit filter in the `audit.xml` file. The system defines 4 default audit filters by default. Use the createAuditFilter command to create additional audit filters in your `audit.xml` configuration file. (String, required)

Return value

The command returns a value of `true` if the system successfully enables the audit filter.

Batch mode example usage

- Using Jython string:

```
AdminTask.enableAuditFilter('-filterRef AuditSpecification_1184689433421')
```

- Using Jython list:

```
AdminTask.enableAuditFilter(['-filterRef', 'AuditSpecification_1184689433421'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.enableAuditFilter('-interactive')
```

getAuditFilter

The getAuditFilter command retrieves the attributes that the system associates with the audit filter specification of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-reference
Specifies the reference ID for an audit filter in your security auditing system configuration. (String, required)

Return value

The command returns a list of attributes for the audit filter specification of interest, as the following sample output displays:

```
{{enabled true}
{name DefaultAuditSpecification_1}
{event SECURITY_AUTHN
SECURITY_AUTHN_MAPPING}
{outcome FAILURE}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditSpecification_1173199825608}
{_Websphere_Config_Data_Type AuditSpecification}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditFilter('-reference AuditSpecification_1173199825608')
```

- Using Jython list:

```
AdminTask.getAuditFilter(['-reference', 'AuditSpecification_1173199825608'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.getAuditFilter('-interactive')
```

getAuditOutcomes

The getAuditOutcomes command retrieves a list of the enabled outcomes for the auditable event type of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventType
Specifies a list of one or more auditable events. To specify a list, separate each outcome with a comma (**,**) character. (String, required)

You can retrieve the event outcome for any of the following auditable events that might be configured in your security auditing system:

Table 54. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved

Table 54. Event types (continued). Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

Return value

The command returns one or multiple outcomes for the event type of interest, as the following sample output displays:

```
SUCCESS
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditOutcomes('-eventType SECURITY_MGMT_PROVISIONING')
```

- Using Jython list:

```
AdminTask.getAuditOutcomes(['-eventType', 'SECURITY_MGMT_PROVISIONING'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.getAuditOutcomes('-interactive')
```

getSupportedAuditEvents

The getSupportedAuditEvents command returns a list of each supported auditable event.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns the following list of possible event outcomes:

```
SECURITY_AUTHN SECURITY_AUTHN_CREDS_MODIFY
SECURITY_AUTHN_DELEGATION SECURITY_AUTHN_MAPPING
SECURITY_AUTHN_TERMINATE SECURITY_AUTHZ
SECURITY_ENCRYPTION SECURITY_MGMT_AUDIT
SECURITY_MGMT_CONFIG SECURITY_MGMT_KEY
```

```
SECURITY_MGMT_POLICY SECURITY_MGMT_PROVISIONING
SECURITY_MGMT_REGISTRY SECURITY_MGMT_RESOURCE
SECURITY_RESOURCE_ACCESS SECURITY_RUNTIME
SECURITY_RUNTIME_KEY SECURITY_SIGNING
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getSupportedAuditEvents()
```

- Using Jython list:

```
AdminTask.getSupportedAuditEvents()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getSupportedAuditEvents('-interactive')
```

getSupportedAuditOutcomes

The getSupportedAuditOutcomes command retrieves a list of each supported outcome for the auditable event filters.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns the following list of possible event outcomes:

```
SUCCESS
INFO
WARNING
ERROR
DENIED
REDIRECT
FAILURE
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getSupportedAuditOutcomes()
```

- Using Jython list:

```
AdminTask.getSupportedAuditOutcomes()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getSupportedAuditOutcomes('-interactive')
```

isAuditFilterEnabled

The isAuditFilterEnabled command determines if the audit filter of interest is enabled in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventType

Specifies a list of one or more auditable events. To specify a list, separate each outcome with a comma (,) character. (String, required)

The following auditable events might be configured in your security auditing system:

Table 55. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

-outcome

Specifies the event outcome. Valid outcomes include `SUCCESS`, `FAILURE`, `REDIRECT`, `ERROR`, `DENIED`, `WARNING`, and `INFO`. (String, required)

Return value

The command returns a value of `true` if the event type of interest is enabled in your configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.isAuditFilterEnabled('-eventType SECURITY_MGMT_PROVISIONING
-outcome SUCCESS')
```

- Using Jython list:

```
AdminTask.isAuditFilterEnabled(['-eventType', 'SECURITY_MGMT_PROVISIONING',
'-outcome', 'SUCCESS'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.isAuditFilterEnabled('-interactive')
```

isEventEnabled

The isEventEnabled command determines if the system enabled at least one audit outcome for the event of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-eventType
Specifies a list of one or more auditable events. To specify a list, separate each outcome with a comma (**,**) character. (String, required)

The following auditable events are available to configure in your security auditing system:

Table 56. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

Return value

The command returns a value of `true` if the audit filter of interest has at least one outcome configured in the `audit.xml` file.

Batch mode example usage

- Using Jython string:

```
AdminTask.isEventEnabled('-eventType SECURITY_AUTHN')
```

- Using Jython list:

```
AdminTask.isEventEnabled(['-eventType', 'SECURITY_AUTHN'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isEventEnabled('-interactive')
```

listAuditFilters

The listAuditFilters command lists each audit filter and the corresponding attributes that the system defines in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of audit filters and the corresponding attributes, as the following example displays:

```
{{enabled true}
{name DefaultAuditSpecification_1}
{event SECURITY_AUTHN SECURITY_AUTHN_MAPPING}
{outcome FAILURE}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1173199825608}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1173199825608}}
{{enabled true}
{name DefaultAuditSpecification_2}
{event {}}
{outcome FAILURE}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1173199825609}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1173199825609}}
{{enabled true}
{name DefaultAuditSpecification_3}
{event SECURITY_RESOURCE_ACCESS}
{outcome FAILURE}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1173199825610}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1173199825610}}
{{enabled true}
{name DefaultAuditSpecification_4}
{event SECURITY_AUTHN_TERMINATE}
{outcome FAILURE}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1173199825611}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1173199825611}}
{{enabled true}
{name myfilter}
{event SECURITY_AUTHZ}
{outcome REDIRECT}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1184365235250}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1184365235250}}
{{enabled true}
{name myfilter1}
{event SECURITY_AUTHZ SECURITY_RESOURCE_ACCESS}
{outcome REDIRECT INFO}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1184365353218}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1184365353218}}
{{enabled true}
{name myfilter}
{event SECURITY_AUTHN SECURITY_AUTHZ}
{outcome SUCCESS INFO}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1184598886859}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1184598886859}}
{{enabled false}
{name myfilter}
{event SECURITY_MGMT_PROVISIONING SECURITY_MGMT_POLICY}
{outcome SUCCESS}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#AuditSpecification_1184689433421}
{_Websphere_Config_Data_Type AuditSpecification}
{filterRef AuditSpecification_1184689433421}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditFilters()
```

- Using Jython list:

```
AdminTask.listAuditFilters()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditFilters('-interactive')
```

listAuditFiltersByEvent

The listAuditFiltersByEvent command retrieves a list of events and event outcomes for each audit filter that is configured in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of events and event outcomes for the audit filters of interest, as the following sample output displays:

```
{AuditSpecification_1173199825608 SECURITY_AUTHN:FAILURE}{AuditSpecification_117
3199825608 SECURITY_AUTHN_MAPPING:FAILURE}{AuditSpecification_1173199825610 SECU
RITY_RESOURCE_ACCESS:FAILURE}{AuditSpecification_1173199825611 SECURITY_AUTHN_TE
RMINATE:FAILURE}{AuditSpecification_1184365235250 SECURITY_AUTHZ:REDIRECT}{Audit
Specification_1184365353218 SECURITY_AUTHZ:REDIRECT;SECURITY_AUTHZ:INFO}{AuditSp
ecification_1184365353218 SECURITY_RESOURCE_ACCESS:REDIRECT;SECURITY_RESOURCE_AC
CESS:INFO}{AuditSpecification_1184598886859 SECURITY_AUTHN:SUCCESS;SECURITY_AUTH
N:INFO}{AuditSpecification_1184598886859 SECURITY_AUTHZ:SUCCESS;SECURITY_AUTHZ:I
NFO}{AuditSpecification_1184689433421 SECURITY_MGMT_PROVISIONING:SUCCESS}{AuditS
pecification_1184689433421 SECURITY_MGMT_POLICY:SUCCESS}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditFiltersByEvent()
```

- Using Jython list:

```
AdminTask.listAuditFiltersByEvent()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditFiltersByEvent('-interactive')
```

listAuditFiltersByRef

The listAuditFiltersByRef command lists all reference ids that correspond to the audit filters that are defined in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of each reference that exists in the audit.xml configuration file, as the following sample output displays:

```
AuditSpecification_1173199825608 AuditSpecification_1173199825609
AuditSpecification_1173199825610 AuditSpecification_1173199825611
AuditSpecification_1184365235250 AuditSpecification_1184365353218
AuditSpecification_1184598886859 AuditSpecification_1184689433421
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditFiltersByRef()
```

- Using Jython list:

```
AdminTask.listAuditFiltersByRef()
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.listAuditFiltersByRef('-interactive')
```

modifyAuditFilter

The modifyAuditFilter command modifies the audit filter specification in the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-filterRef
Specifies the reference ID for the audit filter to modify in your security auditing system configuration. (String, required)

Optional parameters

-name
Specifies a unique name to associate with the audit event filter. (String, optional)

-eventType
Specifies a comma-separated list of one or more event types. (String, optional)

-outcome
Specifies a comma-separated list of one or multiple event outcomes. For each audit event type, you must specify an outcome. Valid outcomes include `SUCCESS`, `FAILURE`, `REDIRECT`, `ERROR`, `DENIED`, `WARNING`, and `INFO`. (String, optional)

-enableFilter
Specifies whether to enable the filter. Specify `true` to enable the filter, or `false` to disable the filter. (Boolean, optional).

Return value

The command returns a value of `true` if the system successfully updates the audit filter.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditFilter('-filterRef AuditSpecification_1173199825608 -name
myname -eventType SECURITY_AUTHN -outcome SUCCESS -enableFilter true')
```

- Using Jython list:

```
AdminTask.modifyAuditFilter(['-filterRef', 'AuditSpecification_1173199825608', '-name',
'myname', '-eventType', 'SECURITY_AUTHN', '-outcome', 'SUCCESS', '-enableFilter',
'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditFilter('-interactive')
```

AuditNotificationCommands command group for the AdminTask object

You can use the Jython scripting language to manage the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditNotificationCommands group to configure and manage audit notifications and audit notification monitors.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Use the following commands to configure your security auditing system notifications:

createAuditNotification

The createAuditNotification command creates an audit notification object in the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-notificationName
Specifies the unique name to assign the audit notification object. (String, required)

-logToSystemOut
Specifies whether the system logs notifications to the `SystemOut.log` file.(Boolean, required)

-sendEmail
Specifies whether to email security auditing subsystem failure notifications. (Boolean, required)

Optional parameters

-emailList
Specifies the email list to send security auditing subsystem failure notifications. (String, optional)

-emailFormat
Specifies the email format. Specify `HTML` for HTML format or `TEXT` for text format. (String, optional)

Return value

The command returns the shortened reference ID of the new audit notification object, as the following sample output displays:

```
WSNotification_1184690835390
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditNotification('[-notificationName mynotification -logToSystemOut
true -sendEmail true -emailList admin@mycompany.com(smtp-server.mycompany.com)
-emailFormat HTML]')
```

- Using Jython list:

```
AdminTask.createAuditNotification(['-notificationName', 'mynotification', '-logToSystemOut',
'true', '-sendEmail', 'true', '-emailList', 'admin@mycompany.com(smtp-server.mycompany.com)',
'-emailFormat', 'HTML'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditNotification('-interactive')
```

createAuditNotificationMonitor

The createAuditNotificationMonitor command creates an audit notification monitor object for the security auditing system. This object monitors the security auditing subsystem for possible failure.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-monitorName
Specifies the unique name of the audit notification monitoring object. (String, required)

-notificationRef
Specifies the reference ID of the audit notification object. (String, required)

-enable
Specifies whether to enable the audit notification monitor. (Boolean, required)

Return value

The command returns the shortened form of the reference ID for the audit notification monitor, as the following sample output displays:

```
AuditNotificationMonitor_1184695615171
```

Batch mode example usage

- Using Jython string:

```
AdminTask.createAuditNotificationMonitor('-monitorName mymonitor -notificationRef
WSNotification_1184690835390 -enable true')
```

- Using Jython list:

```
AdminTask.createAuditNotificationMonitor(['-monitorName', 'mymonitor', '-notificationRef',
'WSNotification_1184690835390', '-enable', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createAuditNotificationMonitor('-interactive')
```

deleteAuditNotification

The deleteAuditNotification command deletes an audit notification object from the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-notificationRef
Specifies the reference ID of the audit notification object to delete. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit notification object from the `audit.xml` configuration file.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditNotification('-notificationRef WSNotification_1184690835390')
```

- Using Jython list:

```
AdminTask.deleteAuditNotification(['-notificationRef', 'WSNotification_1184690835390'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditNotification('-interactive')
```

deleteAuditNotificationMonitorByName

The deleteAuditNotificationMonitorByName command deletes the audit notification monitor that the user specifies with the unique name.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-monitorName
Specifies the unique name of the audit notification monitor to delete. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit notification monitor from the configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditNotificationMonitor('-monitorName mymonitor')
```

- Using Jython list:

```
AdminTask.deleteAuditNotificationMonitor(['-monitorName', 'mymonitor'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditNotificationMonitor('-interactive')
```

deleteAuditNotificationMonitorByRef

The deleteAuditNotificationMonitorByRef command deletes the audit notification monitor that the user specifies with the reference ID.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-monitorRef
Specifies the reference ID of the audit notification monitor object to delete. (String, required)

Return value

The command returns a value of `true` if the system successfully deletes the audit notification monitor of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAuditNotificationMonitor('-monitorRef
AuditNotificationMonitor_1184695615171')
```

- Using Jython list:

```
AdminTask.deleteAuditNotificationMonitor(['-monitorRef',
'AuditNotificationMonitor_1184695615171'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAuditNotificationMonitor('-interactive')
```

getAuditNotification

The getAuditNotification command retrieves the attributes for an audit notification object of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-notificationRef
Specifies the reference ID of the audit notification object of interest. (String, required)

Return value

The command returns a list of attributes for the specific audit notification object, as the following sample output displays:

```
{{name mynotification}
{sslConfig {}}
{logToSystemOut true}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#WSNotification_1184690835390}
{emailList sweetshadow@us.ibm.com(smtp-server.us.ibm.com)}
{sendEmail true}
{_Websphere_Config_Data_Type WSNotification}
{properties {}}
{emailFormat HTML}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditNotification('-notificationRef WSNotification_1184690835390')
```

- Using Jython list:

```
AdminTask.getAuditNotification(['-notificationRef', 'WSNotification_1184690835390'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditNotification('-interactive')
```

getAuditNotificationMonitor

The getAuditNotificationMonitor command retrieves the attributes that the system associates with the audit notification monitor of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-monitorRef
Specifies the reference ID of the audit notification monitor of interest. (String, required)

Return value

The command returns a list of attributes for the audit notification monitor of interest, as the following sample output displays:

```
{{name mymonitor}
{enabled true}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditNotificationMonitor_1184695615171}
{_Websphere_Config_Data_Type AuditNotificationMonitor}
{wsNotification mynotification(cells/Node04Cell|audit.xml#WSNotification_1184690835390)}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditNotificationMonitor('-monitorRef
AuditNotificationMonitor_1184695615171')
```

- Using Jython list:

```
AdminTask.getAuditNotificationMonitor(['-monitorRef',
'AuditNotificationMonitor_1184695615171'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditNotificationMonitor('-interactive')
```

getEmailList

The getEmailList command retrieves the email distribution list for the audit notification object. If the notification monitor is not configured, the audit notification object is not active and the command returns a null value.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns this email list for the active audit notification object, as the following sample output displays:

```
admin@mycompany.com(smtp-server.mycompany.com)
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getEmailList()
```

- Using Jython list:

```
AdminTask.getEmailList()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEmailList('-interactive')
```

getSendEmail

The getSendEmail command displays whether or not the audit notification object sends an email if the audit subsystem fails. If the notification monitor is not configured, the audit notification object is not active and the command returns a null value.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system is configured to send an email to the distribution list.

Batch mode example usage

- Using Jython string:

```
AdminTask.getSendEmail()
```

- Using Jython list:

```
AdminTask.getSendEmail()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getSendEmail('-interactive')
```

getAuditNotificationRef

The getAuditNotificationRef command retrieves the reference ID for the active audit notification object. If the notification monitor is not configured, the audit notification object is not active and the command returns a null value.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns the reference ID of the audit notification object if it is active, as the following sample output displays:

```
WSNotification_1184690835390
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditNotificationRef()
```

- Using Jython list:

```
AdminTask.getAuditNotificationRef()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditNotificationRef('-interactive')
```

getAuditNotificationName

The getAuditNotificationName command retrieves the unique name for the active audit notification object. If the notification monitor is not configured, the audit notification object is not active and the command returns a null value.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns the unique name of the audit notification object, as the following sample output displays:

```
mynotification
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditNotificationName()
```

- Using Jython list:

```
AdminTask.getAuditNotificationName()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditNotificationName('-interactive')
```

isSendEmailEnabled

The isSendEmailEnabled command determines if the system is configured to send an email if the security auditing subsystem fails.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if email notification is enabled.

Batch mode example usage

- Using Jython string:

```
AdminTask.isSendEmailEnabled()
```

- Using Jython list:

```
AdminTask.isSendEmailEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isSendEmailEnabled('-interactive')
```

isAuditNotificationEnabled

The isAuditNotificationEnabled command determines whether the security auditing system notifications are enabled.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if security auditing system notifications are enabled.

Batch mode example usage

- Using Jython string:

```
AdminTask.isAuditNotificationEnabled()
```

- Using Jython list:

```
AdminTask.isAuditNotificationEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isAuditNotificationEnabled()
```

listAuditNotifications

The listAuditNotifications command retrieves the attributes for each audit notification object that is configured in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes for each configured audit notification object, as the following sample output displays:

```
{{name mynotification}
{sslConfig {}}
{logToSystemOut true}
{_Websphere_Config_Data_Id cells/CHEYENNENode04Cell|audit.xml#WSNotification_1184690835390}
{emailList sweetshadow@us.ibm.com(smtp-server.us.ibm.com)}
{sendEmail true}
{notificationRef WSNotification_1184690835390}
{_Websphere_Config_Data_Type WSNotification}
{properties {}}
{emailFormat HTML}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditNotifications()
```

- Using Jython list:

```
AdminTask.listAuditNotifications()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditNotifications('-interactive')
```

listAuditNotificationMonitors

The listAuditNotificationMonitors command lists the attributes for the audit notification monitor that is configured in the `audit.xml` file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes for the audit notification monitor, as the following sample output displays:

```
{{name mymonitor}
{enabled true}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditNotificationMonitor_1184695615171}
```

```
{_Websphere_Config_Data_Type AuditNotificationMonitor}
{monitorRef AuditNotificationMonitor_1184695615171}
{wsNotification mynotification(cells/Node04Cell|audit.xml#WSNotification_1184690835390)}
{notificationRef WSNotification_1184690835390}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.listAuditNotificationMonitors()
```

- Using Jython list:

```
AdminTask.listAuditNotificationMonitors()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAuditNotificationMonitors('-interactive'b)
```

modifyAuditNotification

The modifyAuditNotification command edits the audit notification object in the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-notificationRef
Specifies the reference ID of the audit notification object to edit. (String, required)

Optional parameters

-logToSystemOut
Specifies whether to log notifications to the `SystemOut.log` file. (Boolean, optional)

-sendEmail
Specifies whether to email notifications. (Boolean, optional)

-emailList
Specifies the email address of distribution list where the system sends email notifications. (String, optional)

-emailFormat
Specifies the email format. Specify `HTML` for HTML format or `TEXT` for text format. (String, optional)

Return value

The command returns a value of `true` if the system successfully updates the security auditing system configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditNotification('-notificationRef WSNotification_1184690835390
-logToSystemOut false -sendEmail true -emailList admin@mycompany.com(smtp-server.mycompany.com)
-emailFormat TEXT')
```

- Using Jython list:

```
AdminTask.modifyAuditNotification(['-notificationRef', 'WSNotification_1184690835390',
'-logToSystemOut', 'false', '-sendEmail', 'true', '-emailList',
'admin@mycompany.com(smtp-server.mycompany.com)', '-emailFormat', 'TEXT'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditNotification('-interactive')
```

modifyAuditNotificationMonitor

The modifyAuditNotificationMonitor command edits the audit notification monitor configuration for the security auditing system.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-monitorRef
Specifies the reference ID of the audit notification monitor of interest. (String, required)

Optional parameters

-notificationRef
Specifies the reference ID of the audit notification object. (String, optional)

-enable
Specifies whether to enable the audit notification monitor. (Boolean, optional)

Return value

The command returns a value of `true` if the system successfully updates the audit notification monitor configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditNotificationMonitor('-monitorRef AuditNotificationMonitor_1184695615171
-notificationRef WSNotification_1184690835390 -enable true')
```

- Using Jython list:

```
AdminTask.modifyAuditNotificationMonitor(['-monitorRef', 'AuditNotificationMonitor_1184695615171',
'-notificationRef', 'WSNotification_1184690835390', '-enable', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditNotificationMonitor('-interactive')
```

setEmailList

The setEmailList command specifies the distribution list to send email notifications to if the security auditing subsystem fails.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-emailList
Specifies the email address or email distribution list to send audit notifications to. (String, required)

Return value

The command returns a value of `true` if the system successfully sets the email notification list for the notification object.

Batch mode example usage

- Using Jython string:

```
AdminTask.setEmailList('[-emailList admin@mycompany.com(smtp-server.mycompany.com)]')
```

- Using Jython list:

```
AdminTask.setEmailList(['-emailList', 'admin@mycompany.com(smtp-server.mycompany.com)'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.setEmailList('-interactive')
```

setSendEmail

The setSendEmail command enables or disables email notifications for the security auditing system.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-enable
Specifies whether to enable the system to send audit notifications by email. (Boolean, required)

Return value

The command returns a value of `true` if the system successfully modifies the configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.setSendEmail('-enable true')
```

- Using Jython list:

```
AdminTask.setSendEmail(['-enable', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setSendEmail('-interactive')
```

AuditPolicyCommands command group for the AdminTask object

You can use the Jython scripting language to manage the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditPolicyCommands group to enable and configure the security auditing system.

Use the following commands to configure, query, and manage the security auditing system:

- “disableAudit” on page 366
- “disableVerboseAudit” on page 366

disableAudit

The disableAudit command disables security auditing in the `audit.xml` configuration file.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully disables security auditing.

Batch mode example usage

- Using Jython string:

```
AdminTask.disableAudit()
```

- Using Jython list:

```
AdminTask.disableAudit()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.disableAudit('-interactive')
```

disableVerboseAudit

The disableVerboseAudit command disables the verbose capture of audit data for the security auditing system.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully disables the verbose capture of audit data.

Batch mode example usage

- Using Jython string:

```
AdminTask.disableVerboseAudit()
```

- Using Jython list:

```
AdminTask.disableVerboseAudit()
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.disableVerboseAudit('-interactive')
```

enableAudit

The enableAudit command enables security auditing in the `audit.xml` configuration file. By default, security auditing is disabled.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully enables security auditing.

Batch mode example usage

- Using Jython string:

```
AdminTask.enableAudit()
```

- Using Jython list:

```
AdminTask.enableAudit()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.enableAudit('-interacive')
```

enableVerboseAudit

The enableVerboseAudit command sets the security auditing system to perform verbose capture of audit data.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully sets the security auditing system to perform verbose capture of audit data.

Batch mode example usage

- Using Jython string:

```
AdminTask.enableVerboseAudit()
```

- Using Jython list:

```
AdminTask.enableVerboseAudit()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.enableVerboseAudit('-interactive')
```

getAuditPolicy

The getAuditPolicy command retrieves each attribute that is associated with the audit policy in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a list of attributes for the security auditing system, as the following sample output displays:

```
{{auditEventFactories {{{name auditEventFactoryImpl_1}
{properties {}}
{className com.ibm.ws.security.audit.AuditEventFactoryImpl}
{auditServiceProvider auditServiceProviderImpl_1(cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608)}
{auditSpecifications DefaultAuditSpecification_1(cells/Node04Cell|audit.xml#AuditSpecification_1173199825608)
DefaultAuditSpecification_2(cells/Node04Cell|audit.xml#AuditSpecification_1173199825609)
DefaultAuditSpecification_3(cells/Node04Cell|audit.xml#AuditSpecification_1173199825610)
DefaultAuditSpecification_4(cells/Node04Cell|audit.xml#AuditSpecification_1173199825611)}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditEventFactory_1173199825608}
{_Websphere_Config_Data_Type AuditEventFactory}}}}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditPolicy_1173199825608}
{auditServiceProviders {{{auditSpecifications
DefaultAuditSpecification_1(cells/Node04Cell|audit.xml#AuditSpecification_1173199825608)
DefaultAuditSpecification_2(cells/Node04Cell|audit.xml#AuditSpecification_1173199825609)
DefaultAuditSpecification_3(cells/Node04Cell|audit.xml#AuditSpecification_1173199825610)
DefaultAuditSpecification_4(cells/Node04Cell|audit.xml#AuditSpecification_1173199825611)}
{name auditServiceProviderImpl_1}
{_Websphere_Config_Data_Id cells/Node04Cell|audit.xml#AuditServiceProvider_1173199825608}
{maxFileSize 1}
{_Websphere_Config_Data_Type AuditServiceProvider}
{fileLocation ${PROFILE_ROOT}/logs/server1}
{className com.ibm.ws.security.audit.BinaryEmitterImpl}
{properties {}}
{eventFormatterClass {}}
{maxLogs 100}}}}
{securityXmlSignerCertAlias auditSignCert}
{properties {}}
{securityXmlSignerScopeName (cell):Node04Cell:(node):Node04}
{auditorPwd SweetShadowsPwd}
{_Websphere_Config_Data_Type AuditPolicy}
{securityXmlSignerKeyStoreName NodeDefaultSignersStore}
{verbose false}
{auditPolicy WARN}
{encrypt false}
{managementScope {}}
{encryptionCert {}}
{batching false}
{auditorId SweetShadow}
{auditEnabled false}
{sign true}}
```

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditPolicy()
```

- Using Jython list:

```
AdminTask.getAuditPolicy()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditPolicy('-interactive')
```

getAuditSystemFailureAction

The getAuditSystemFailureAction command displays the action that the application server takes if a failure occurs in the security auditing subsystem.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a string that describes the action that the application server takes if the security auditing subsystem fails. Possible values are `WARN`, `NOWARN`, or `FATAL`.

Table 57. Application server actions if the security auditing subsystem fails. The following table describes the behavior associated with each action that the application server takes if the security auditing subsystem fails:

WARN	Specifies that the application server should notify the auditor, stop security auditing, and continue to run the application server process.
NOWARN	Specifies that the application server should not notify the auditor, but should stop security auditing and continue to run the application server process
FATAL	Specifies that the application server should notify the auditor, stop security auditing, and stop the application server process.

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditSystemFailureAction()
```

- Using Jython list:

```
AdminTask.getAuditSystemFailureAction()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditSystemFailureAction('-interactive')
```

getAuditorId

The getAuditorId command retrieves the name of the user who is assigned as the auditor.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns the name of the user who is assigned as the auditor.

Batch mode example usage

- Using Jython string:

```
AdminTask.getAuditorId()
```

- Using Jython list:

```
AdminTask.getAuditorId()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getAuditorId('-interactive')
```

isAuditEnabled

The isAuditEnabled command determines whether the security auditing is enabled in your configuration. By default, auditing is not enabled in the `audit.xml` configuration file.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if security auditing is enabled in your environment. If the command returns a value of `false`, security auditing is disabled.

Batch mode example usage

- Using Jython string:

```
AdminTask.isAuditEnabled()
```

- Using Jython list:

```
AdminTask.isAuditEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isAuditEnabled('-interactive')
```

isVerboseAuditEnabled

The isVerboseAuditEnabled command determines whether or not the security auditing system verbosely captures audit data.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the security auditing system is configured to verbosely capture audit data.

Batch mode example usage

- Using Jython string:

```
AdminTask.isVerboseAuditEnabled()
```

- Using Jython list:

```
AdminTask.isVerboseAuditEnabled()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.isVerboseAuditEnabled('-interactive')
```

mapAuditGroupIDsOfAuthorizationGroup

The mapAuditGroupIDsOfAuthorizationGroup command maps the special subjects to users in the registry.

The user must have the monitor administrative role to run this command.

Target object

None.

Return value

The command does not return output.

Batch mode example usage

- Using Jython string:

```
AdminTask.mapAuditGroupIDsOfAuthorizationGroup()
```

- Using Jython list:

```
AdminTask.mapAuditGroupIDsOfAuthorizationGroup()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.mapAuditGroupIDsOfAuthorizationGroup('-interactive')
```

modifyAuditPolicy

The modifyAuditPolicy command modifies the audit policy attributes in the `audit.xml` configuration file. You can use this command to modify one or multiple attributes.

The user must have the auditor administrative role to run this command.

Target object

None.

Optional parameters

-auditEnabled
Specifies whether security auditing is enabled in your configuration. (Boolean, optional)

-auditPolicy
Specifies the action that the application server takes if the security auditing subsystem fails. (String, optional)

Table 58. auditPolicy parameter values. The following table describes the valid values for the auditPolicy parameter:

WARN	Specifies that the application server should notify the auditor, stop security auditing, and continue to run the application server process.
NOWARN	Specifies that the application server should not notify the auditor, but should stop security auditing and continue to run the application server process
FATAL	Specifies that the application server should notify the auditor, stop security auditing, and stop the application server process.

-auditorId
Specifies the name of the user that the system assigns as the auditor. (String, optional)

-auditorPwd
Specifies the password for the auditor id. (String, optional)

-sign
Specifies whether to sign audit records. Use the AuditSigningCommands command group to configure signing settings. (Boolean, optional)

-encrypt
Specifies whether to encrypt audit records. Use the AuditEncryptionCommands command group to configure encryption settings. (Boolean, optional)

-verbose
Specifies whether to capture verbose audit data. (Boolean, optional)

-encryptionCert
Specifies the reference ID of the certificate to use for encryption. Specify this parameter if you set the -encrypt parameter to `true`. (String, optional)

Return value

The command returns a value of `true` if the system successfully updates the security auditing system policy.

Batch mode example usage

- Using Jython string:

```
AdminTask.modifyAuditPolicy('-auditEnabled true -auditPolicy NOWARN -auditorId
testuser -auditorPwd testuserpwd -sign false -encrypt false -verbose false')
```

- Using Jython list:

```
AdminTask.modifyAuditPolicy(['-auditEnabled', 'true', '-auditPolicy', 'NOWARN',
'-auditorId', 'testuser', '-auditorPwd', 'testuserpwd', '-sign', 'false', '-encrypt',
'false', '-verbose', 'false'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.modifyAuditPolicy('-interactive')
```

setAuditSystemFailureAction

The setAuditSystemFailureAction command sets the action that the application server takes if the security auditing subsystem fails.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-action
Specifies the action to take if the security auditing subsystem fails. (String, required)

Table 59. Action parameters. The following table describes the valid values for the action parameter:

WARN	Specifies that the application server should notify the auditor, stop security auditing, and continue to run the application server process.
NOWARN	Specifies that the application server should not notify the auditor, but should stop security auditing and continue to run the application server process

Table 59. Action parameters (continued). The following table describes the valid values for the action parameter:

FATAL	Specifies that the application server should notify the auditor, stop security auditing, and stop the application server process.

Return value

The command returns a value of `true` if the system successfully updates the security auditing system policy.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAuditSystemFailureAction('-action NOWARN')
```

- Using Jython list:

```
AdminTask.setAuditSystemFailureAction(['-action', 'NOWARN'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setAuditSystemFailureAction('-interactive')
```

resetAuditSystemFailureAction

The resetAuditSystemFailureAction command sets the action that the application server takes if the security auditing system fails to the `NOWARN` setting.

The user must have the auditor administrative role to run this command.

Target object

None.

Return value

The command returns a value of `true` if the system successfully updates your configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.resetAuditSystemFailureAction()
```

- Using Jython list:

```
AdminTask.resetAuditSystemFailureAction()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.resetAuditSystemFailureAction('-interactive')
```

setAuditorId

The setAuditorId command sets the name of the user to assign as the auditor.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-name
Specifies the name of the user to assign as the auditor. (String, required)

Return value

The command returns a value of `true` if the system successfully updates your configuration.

Batch mode example usage

- Using Jython string:

```
AdminTAsk.setAuditorId('-name myAdmin')
```

- Using Jython list:

```
AdminTAsk.setAuditorId(['-name', 'myAdmin'])
```

Interactive mode example usage

- Using Jython string:

```
AdminTAsk.setAuditorId('-interactive')
```

setAuditorPwd

The setAuditorPwd command sets the password for the auditor.

The user must have the auditor administrative role to run this command.

Target object

None.

Required parameters

-password
Specifies the password for the user assigned as the auditor. (String, required)

Return value

The command returns a value of `true` if the system successfully updates your configuration.

Batch mode example usage

- Using Jython string:

```
AdminTask.setAuditorPwd('-password myAdminPassword')
```

- Using Jython list:

```
AdminTask.setAuditorPwd(['-password', 'myAdminPassword'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setAuditorPwd('-interactive')
```

AuditEventFormatterCommands command group for the AdminTask object

You can use the Jython scripting language to manage the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditEventFormatterCommands group to manage the event formatter for the audit service provider.

Use the following commands to display information about the audit event formatter:

- “getEventFormatterClass” on page 375

getEventFormatterClass

The getEventFormatterClass command retrieves the class name of the event formatter specified by the reference ID of the binary file audit service provider of interest.

The user must have the monitor administrative role to run this command.

Target object

None.

Required parameters

-emitterRef
Specifies the reference ID of the audit service provider implementation of interest. (String, required)

Return value

The command returns a null value if the event formatter class is not defined for the audit service provider implementation of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.getEventFormatterClass('-emitterRef AuditServiceProvider_1173199825608')
```

- Using Jython list:

```
AdminTask.getEventFormatterClass(['-emitterRef', 'AuditServiceProvider_1173199825608'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getEventFormatterClass('-interactive')
```

AuditReaderCommands command group for the AdminTask object

You can use the Jython scripting language to manage the security auditing system with the wsadmin tool. Use the commands and parameters in the AuditReaderCommands group to display audit record information from the binary audit log.

Use the following commands to query the binary audit log:

- "binaryAuditLogReader"
- "showAuditLogEncryptionInfo" on page 378

binaryAuditLogReader

The binaryAuditLogReader command reads the default binary audit log and generates an HTML report based on the parameters you provide. You must use the auditor security role to use this command.

Target object

None.

Required parameters

-fileName
Specifies the fully qualified file name for the binary audit log. (String, required)

-outputLocation
Specifies the location of the HTML report that the command generates. (String, required)

Optional parameters

-reportMode
Specifies the type of report to generate. Valid values include `basic`, `complete`, or `custom`. The `basic` report provides the following configuration information:

- creationTime
- action
- progName
- registryType
- domain
- realm
- remoteAddr
- remotePort
- remoteHost
- resourceName
- resourceType
- resourceUniqueId

The `complete` report provides the data included by the default report type and each additional datapoint of interest. The `custom` report allows you to specify only the datapoints you choose to see generated in the report. The default value is basic. (String, optional)

See the Data point values table for the information that is available with each of the report types.

-eventFilter
Specifies the audit types to read and report. Specify one or more audit event types. If you specify more than one value for the eventFilter parameter, separate each audit event type with a colon character (`:`). (String, optional)

-outcomeFilter
Specifies the audit event outcomes to read and report. Specify one or more audit event outcomes. If you specify more than one value for the outcomeFilter parameter, separate each audit event outcome with a colon character (`:`). (String, optional)

-sequenceFilter
Specifies a list of beginning and ending sequence numbers. Use the `a:b` syntax, where `a`, the starting sequence number where the HTML report begins, and is less than or equal to `b`, the sequence number where the HTML report ends. A single sequence may also be specified, such as `-sequenceFilter 10`, to only generate a report for the tenth record. (String, optional)

-timeStampFilter
Specifies the time stamp range of records to read and report. Use the `a:b` syntax, where `a` and `b` are strings in the format `java.text.SimpleDateFormat("MMddhhmmyyyy")`. You can also specify a single timestamp. (String, optional)

-keyStorePassword
Specifies password to open the keystore. (String, optional)

-dataPoints
Specifies a list of specific audit data to use to generate the report. Use this option only when you set the reportMode parameter as `custom`. If you specify multiple data points, separate each data point with a colon character (`:`). (String, optional)

Table 60. Data point values. The following table includes the available data points, the report mode, its context object name, its field name, and a description of the information:

Data point name	reportMode value	Context object name	Field name	Description
RemoteAddr	basic	SessionContextObj	remoteAddr	The data point provides the IP address for the default remote host.

Table 60. Data point values (continued). The following table includes the available data points, the report mode, its context object name, its field name, and a description of the information:

Data point name	reportMode value	Context object name	Field name	Description
RemotePort	basic	SessionContextObj	remotePort	The data point provides the port of the default remote host.
RemoteHost	basic	SessionContextObj	remoteHost	The data point provides the host name of the remote host.
RegistryType	basic	RegistryContextObj	type	The data point provides the type of user registry that is being used, such as Lightweight Directory Access Protocol (LDAP) or AIX®.
Domain	basic	ProcessContextObj	domain	The data point provides the domain to which the user belongs.
Realm	basic	ProcessContextObj	realm	The data point provides the registry partition to which the user belongs.
CreationTime	basic	EventContextObj	creationTime	The data point provides the date an event was created.
ProgName	basic	AccessContextObj	progName	The data point provides the name of the program that was involved in the event.
Action	basic	AccessContextObj	action	The data point provides the action being performed.
ResourceName	basic	AccessContextObj	resourceName	The data point provides the name of the resource in the context of the application.
ResourceType	basic	AccessContextObj	resourceType	The data point provides the type of resource.
ResourceUniqueId	basic	AccessContextObj	resourceUniqueId	The data point provides the unique identifier of the resource.
SessionId	complete	SessionContextObj	sessionId	The data point provides an identifier for the default user session.
FirstCaller	basic	PropagationContextObj	firstCaller	The data point provides the identity of the first user in the caller list.
DelegationType	complete	DelegationContextObj	delegationType	The data point provides the delegation type. The delegation types are no delegation, simple delegation, method delegation or switch user delegation information.
RoleName	complete	DelegationContextObj	roleName	The data point provides the Run as role that is being used. The Run as roles are runAsClient, runAsSpecified, runAsSystem, or own ID.
IdentityName	complete	DelegationContextObj	identityName	The data point provides information about the mapped user.
AuthnType	complete	AuthnContextObj	authnType	The data point provides the type of authentication that is being used.
Provider	complete	ProviderContextObj	provider	The data point returns the provider of the authentication or authorization service.
ProviderStatus	complete	ProviderContextObj	providerStatus	The data point provides the status of whether the authentication or authorization event was successfully processed by the provider.
MappedSecurityDomain	complete	AuthnMappingContextObj	mappedSecurityDomain	The data point provides the security domain after the mapping has occurred.
MappedRealm	complete	AuthnMappingContextObj	mappedRealm	The data point provides the realm name after the mapping has occurred.
MappedUserName	complete	AuthnMappingContextObj	mappedUserName	The data point provides the user name after the mapping has occurred.
TerminateReason	basic	AuthnTermContextObj	terminateReason	The data point provides the reason that authentication ended.
RegistryUserName	basic	AccessContextObj	registryUserName	The data point provides the name of the user in the registry.
AppUserName	basic	AccessContextObj	appUserName	The data point provides the name of the user within an application.
AccessDecision	complete	AccessContextObj	accessDecision	The data point provides the decision of the authorization call.
PermissionsChecked	complete	AccessContextObj	permissionsChecked	The data point provides the permissions that were checked during the authorization call.
PermissionsGranted	complete	AccessContextObj	permissionsGranted	The data point provides the permissions that were granted during the authorization call.
RolesChecked	complete	AccessContextObj	rolesChecked	The data point provides the roles that were checked during the authorization call.
RolesGranted	complete	AccessContextObj	rolesGranted	The data point provides the roles that were granted during the authorization call
PolicyName	complete	PolicyContextObj	policyName	The data point provides the name of the policy.
PolicyType	complete	PolicyContextObj	policyType	The data point provides the type of policy.
KeyLabel	basic	KeyContextObj	keyLabel	The data point provides the key or certificate label.
KeyLocation	basic	KeyContextObj	keyLocation	The data point provides the physical location of the key database.
CertLifetime	basic	KeyContextObj	certLifetime	The data point provides the date when a certificate expires.
MgmtType	complete	MgmtContextObj	mgmtType	The data point provides the type of management operation.
MgmtCommand	complete	MgmtContextObj	mgmtCommand	The data point provides the application-specific command that was performed.
Url	complete	ResponseContextObj	url	The data point provides the URL of the HTTP request.
CallerList	basic	PropagationContextObj	callerList	The data point provides a list of names that represent the identities of the users.
HttpRequestHeaders	complete	ResponseContextObj	httpRequestHeaders	The data point provides the HTTP request headers that are provided by the client.
HttpResponseHeaders	complete	ResponseContextObj	httpResponseHeaders	The data point provides the HTTP response headers that are returned by the server
TargetInfoName	complete	ResponseContextObj	httpResponseHeaders	The object the operation is targeted against
TargetInfoUniqueId	complete	ResponseContextObj	httpResponseHeaders	The unique identifier of the target

Table 60. Data point values (continued). The following table includes the available data points, the report mode, its context object name, its field name, and a description of the information:

Data point name	reportMode value	Context object name	Field name	Description
OutcomeReasonCode	complete	ResponseContextObj	httpResponseHeaders	A code mapping to an outcome decision 1 means a certificate parsing error 2 means a security context error

Return value

The command returns the HTML report based on the values specified for each parameter to the location specified by the outputLocation parameter.

Batch mode example usage

- Using Jython string:

```
AdminTask.binaryAuditLogReader('[-fileName myFileName -reportMode basic
-keyStorePassword password123 -outputLocation /binaryLogs]')
```

- Using Jython list:

```
AdminTask.binaryAuditLogReader(['-fileName', 'myFileName', '-reportMode', 'basic',
'-keyStorePassword', 'password123', '-outputLocation', '/binaryLogs'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.binaryAuditLogReader('-interactive')
```

showAuditLogEncryptionInfo

The showAuditLogEncryptionInfo command displays information about the keystore that the auditing system uses to encrypt audit records. Use this information as a hint of the keystore password in order to decrypt encrypted audit logs in the binary audit log.

Target object

None.

Required parameters

-fileName
Specifies the fully qualified path of the binary audit log. (String, required)

Return value

The command returns the certificate alias and the fully qualified path to the keystore of interest.

Batch mode example usage

- Using Jython string:

```
AdminTask.showAuditLogEncryptionInfo('-fileName myFileName')
```

- Using Jython list:

```
AdminTask.showAuditLogEncryptionInfo(['-fileName', 'myFileName'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.showAuditLogEncryptionInfo('-interactive')
```

SSLMigrationCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to migrate key store configurations. Use the commands in the SSLMigrationCommands group to convert self-signed certificates to chained personal certificates and to enable writable key rings.

The SSLMigrationCommands command group for the AdminTask object includes the following commands:

- "convertSelfSignedCertificatesToChained command"
- "enableWritableKeyrings command" on page 380
- "convertSSLConfig command" on page 381

convertSelfSignedCertificatesToChained command

The convertSelfSignedCertificatesToChained command converts specific self-signed certificates to chained personal certificates.

Note: Chained certificates are the default certificate type in Websphere Application Server Version 7.0. The convertSelfSignedCertificatesToChained command takes information from the self-signed certificate—such as issued-to DN, size, and life span—and creates a chained certificate with the same information. The new chained certificate replaces the self-signed certificate. Signer certificates from the self-signed certificate that are distributed across the security configuration are replaced with the signer certificates from the root certificate used to sign the chained certificate.

Syntax

The command has the following syntax:

```
wsadmin>$AdminTask convertSelfSignedCertificatesToChained
                   [-certificateReplacementOption ALL_CERTIFICATES | DEFAULT_CERTIFICATES | KEYSTORE_CERTIFICATES]
                   [-keyStoreName keystore_name]
                   [-keyStoreScope keystore_scope]
                   [-rootCertificateAlias alias_name]
```

Required parameters

certificateReplacementOption

Specifies the convert self-signed certificates replacement options. (String, required)

Specify the value for the parameter as one of the following options:

ALL_CERTIFICATES

This option looks for all self-signed certificates in all keystores with in the specified scope.

The scope can be provided in the -keyStoreScope parameter. If no scope is provided using the -keyStoreScope parameter, all scopes are visited.

DEFAULT_CERTIFICATES

This option looks for self-signed certificates in the default CellDefaultKeyStore and NodeDefaultKeyStore keystores within the specified scope.

The scope can be provided with the -keyStoreScope parameter. If no scope is provided using the -keyStoreScope parameter, all scopes are visited.

KEYSTORE_CERTIFICATES

This option replaces only those self-signed certificates in the keystore that are specified by the -keyStoreName parameter.

If no scope is provided using the -keyStoreScope parameter, the default scope is used.

Optional parameters

keyStoreName
Specifies the name of a keystore in which to look for self-signed certificates to convert. Use this parameter with the KEYSTORE_CERTIFICATES option on the certificateReplacementOption parameter. (String, optional)

keyStoreScope
Specifies the name of the scope in which to look for the self-signed certificates to convert. (String, optional)

rootCertificateAlias
Specifies the root certificate to use from the default root store used to sign the chained certificate. The default value is `root`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask convertSelfSignedCertificatesToChained {-certificateReplacementOption ALL_CERTIFICATES -keyStoreName testKS}
```

- Using Jython string:

```
AdminTask.convertSelfSignedCertificatesToChained('[-certificateReplacementOption ALL_CERTIFICATES -keyStoreName testKS]')
```

- Using Jython list:

```
AdminTask.convertSelfSignedCertificatesToChained(['-certificateReplacementOption', 'ALL_CERTIFICATES', '-keyStoreName', 'testKS'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask exchangeSigners {-interactive}
```

- Using Jython:

```
AdminTask.exchangeSigners('-interactive')
```

enableWritableKeyrings command

The enableWritableKeyrings command modifies the keystore and enables writable SAF support. The system uses this command during migration. The command creates additional writable keystore objects for the control region and servant region key rings for SSL keystores.

Required parameters

-keyStoreName
Specifies the name that uniquely identifies the keystore that you want to delete. (String, required)

Optional parameters

-controlRegionUser
Specifies the control region user to use to enable writable key rings. (String, optional)

-servantRegionUser
Specifies the servant region user to enable writable key rings. (String, optional)

-scopeName
Specifies the name that uniquely identifies the management scope, for example: `(cell):localhostNode01Cell`. (String, optional)

Examples

Batch mode example usage:

- Using Jython string:

```
AdminTask.enableWritableKeyrings('[-keyStoreName testKS -controlRegionUser CRUser1 -servantRegionUser SRUser1]')
```

- Using Jython list:

```
AdminTask.enableWritableKeyrings(['-keyStoreName', 'testKS', '-controlRegionUser', 'CRUser1', '-servantRegionUser', 'SRUser1'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.enableWritablekeyrings('-interactive')
```

convertSSLConfig command

The convertSSLConfig command migrates existing SSL configurations to the new configuration object format for SSL configurations.

Required parameters

-sslConversionOption
Specifies how the system converts the SSL configuration. Specify the CONVERT_SSLCONFIGS value to convert the SSL configuration objects from the previous SSL configuration object to the new SSL configuration object. Specify the CONVERT_TO_DEFAULT value to convert the SSL configuration to a centralized SSL configuration, which also removes the SSL configuration direct referencing from the servers.

Optional parameters

None.

Examples

Batch mode example usage:

- Using Jython string:

```
AdminTask.convertSSLConfig('[-keyStoreName testKS -controlRegionUser CRUser1 -servantRegionUser SRUser1]')
```

- Using Jython list:

```
AdminTask.convertSSLConfig(['-keyStoreName', 'testKS', '-controlRegionUser', 'CRUser1', '-servantRegionUser', 'SRUser1'])
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.convertSSLConfig('-interactive')
```

IdMgrConfig command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure the virtual member manager with the wsadmin tool. The commands and parameters in the IdMgrConfig group can be used to create and manage your entity type configuration.

The IdMgrConfig command group for the AdminTask object includes the following commands:

createIdMgrSupportedEntityType

The **createIdMgrSupportedEntityType** command creates a supported entity type configuration.

Parameters

-name
The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required)

-defaultParent
The default parent node for the supported entity type. (String, required)

-rdnProperties
The RDN® attribute name for the supported entity type in the entity domain name. To reset all values of the rdnProperties parameter, specify a blank string ("") (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrSupportedEntityType {-name entity1
–defaultParent node1 –rdnProperties rdn1}
```

- Using Jython string:

```
AdminTask.createIdMgrSupportedEntityType ('[-name entity1
–defaultParent node1 –rdnProperties rdn1]')
```

- Using Jython list:

```
AdminTask.createIdMgrSupportedEntityType (['-name', 'entity1',
'–defaultParent', 'node1', '–rdnProperties', 'rdn1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrSupportedEntityType {-interactive}
```

- Using Jython string:

```
AdminTask.createIdMgrSupportedEntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createIdMgrSupportedEntityType (['-interactive'])
```

deleteIdMgrSupportedEntityType

The **deleteIdMgr Supported EntityType** command deletes the supported entity type configuration that you specify.

Parameters

-name
The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrSupported EntityType {-name entity1}
```

- Using Jython string:

```
AdminTask.deleteIdMgrSupported EntityType ('[-name entity1]')
```

- Using Jython list:

```
AdminTask.deleteIdMgr SupportedEntityType (['-name', 'entity1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgr SupportedEntityType {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgr SupportedEntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgr SupportedEntityType (['-interactive'])
```

getIdMgrSupportedEntityType

The **getIdMgr Supported EntityType** command returns the configuration of the supported entity type that you specify.

Parameters

-name
The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrSupported EntityType {-name entity1}
```

- Using Jython string:

```
AdminTask.getIdMgrSupported EntityType ('[-name entity1]')
```

- Using Jython list:

```
AdminTask.getIdMgrSupported EntityType (['-name', 'entity1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrSupported EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrSupported EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrSupported EntityType (['-interactive'])
```

isIdMgrUseGlobalSchemaForModel

The **isIdMgrUseGlobalSchemaForModel** command returns a boolean that indicates whether the global schema option is enabled for the data model for the specified domain in a multiple security domain environment.

Parameters

-securityDomainName
The name that uniquely identifies the security domain. (String, required)

Returns

A Boolean value that indicates whether global schema option is enabled for the data model for the specified domain.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask isIdMgrUseGlobalSchemaForModel {–securityDomainName mysecDomain}
```

- Using Jython string:

```
AdminTask.isIdMgrUseGlobalSchemaForModel ('[–securityDomainName mysecDomain]')
```

- Using Jython list:

```
AdminTask.isIdMgrUseGlobalSchemaForModel (['–securityDomainName', 'mysecDomain'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask isIdMgrUseGlobalSchemaForModel {-interactive}
```

- Using Jython string:

```
AdminTask.isIdMgrUseGlobalSchemaForModel ('[-interactive]')
```

- Using Jython list:

```
AdminTask.isIdMgrUseGlobalSchemaForModel (['-interactive'])
```

listIdMgrSupportedEntityTypes

The **listIdMgr Supported EntityTypes** command lists all of the supported entity types that are configured.

Parameters

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Returns

A list that contains the names of the supported entity types

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgr SupportedEntityTypes
```

- Using Jython string:

```
AdminTask.listIdMgr SupportedEntityTypes()
```

- Using Jython list:

```
AdminTask.listIdMgr SupportedEntityTypes()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupportedEntityTypes {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrSupported EntityTypes ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrSupported EntityTypes (['-interactive'])
```

listIdMgrGroupsForRoles

The **listIdMgrGroupsForRoles** command lists the mapping of groups to roles in federated repositories.

Parameters

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Returns

A Map object that contains roleName as the key, and the value of each key is a list of uniqueNames.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrGroupsForRoles
```

- Using Jython string:

```
AdminTask.listIdMgrGroupsForRoles ()
```

- Using Jython list:

```
AdminTask.listIdMgrGroupsForRoles ()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrGroupsForRoles {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrGroupsForRoles ('[interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrGroupsForRoles (['interactive'])
```

listIdMgrUsersForRoles

The **listIdMgrUsersForRoles** command lists the mapping of users to roles in federated repositories.

Parameters

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Returns

A Map object that contains roleName as the key, and the value of each key is a list of uniqueNames.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrUsersForRoles
```

- Using Jython string:

```
AdminTask.listIdMgrUsersForRoles ()
```

- Using Jython list:

```
AdminTask.listIdMgrUsersForRoles ()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrUsersForRoles {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrUsersForRoles ('[-interactive]'))
```

- Using Jython list:

```
AdminTask.listIdMgrUsersForRoles (['-interactive'])
```

mapIdMgrUserToRole

The **mapIdMgrUserToRole** command maps a user to a specified role in federated repositories. You can map a user to only one role.

Parameters

-roleName
The name of the role. Valid values are IdMgrAdmin, IdMgrReader, or IdMgrWriter, which are the federated repositories pre-defined roles. (String, required)

-userId
The user ID or unique name of the user to whom you want to map the specified role. If you specify the user ID, it should correspond to a unique user in the repository. (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask mapIdMgrUserToRole {-roleName IdMgrWriter -userId uid=user1,o=customrealm}
```

- Using Jython string:

```
AdminTask.mapIdMgrUserToRole ('[-roleName IdMgrWriter -userId user1,o=customrealm]')
```

- Using Jython list:

```
AdminTask.mapIdMgrUserToRole (['-roleName', 'IdMgrWriter',
'-userId', 'uid=user1,o=customrealm'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask mapIdMgrUserToRole {-interactive}}
```

- Using Jython string:

```
AdminTask.mapIdMgrUserToRole ('[-interactive]'))
```

- Using Jython list:

```
AdminTask.mapIdMgrUserToRole (['-interactive'])
```

mapIdMgrGroupToRole

The **mapIdMgrGroupToRole** command maps a group to a specified role in federated repositories. You can map a group to only one role.

Parameters

-roleName
The name of the role. Valid values are IdMgrAdmin, IdMgrReader, or IdMgrWriter, which are the federated repositories pre-defined roles. (String, required)

-groupId
The common name or unique name of the group to which you want to map the specified role. If you specify the common name, it should correspond to a unique group in the repository. Alternately, to map all logged-in users to the specified role, you can specify a special subject with the value ALLAUTHENTICATED. (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl (example 1):

```
$AdminTask mapIdMgrGroupToRole {-roleName IdMgrReader
-groupId cn=group1,o=customrealm}
```

Using Jacl (example 2):

```
$AdminTask mapIdMgrGroupToRole {-roleName IdMgrWriter
-groupId ALLAUTHENTICATED}
```

- Using Jython string (example 1):

```
AdminTask.mapIdMgrGroupToRole ('[-roleName IdMgrReader
-groupId cn=group1,o=customrealm]')
```

Using Jython string (example 2):

```
AdminTask.mapIdMgrGroupToRole ('[-roleName IdMgrWriter
-groupId ALLAUTHENTICATED]')
```

- Using Jython list (example 1):

```
AdminTask.mapIdMgrGroupToRole (['-roleName', 'IdMgrReader',
'-groupId', 'cn=group1,o=customrealm'])
```

Using Jython list (example 2):

```
AdminTask.mapIdMgrGroupToRole (['-roleName', 'IdMgrReader',
'-groupId', 'ALLAUTHENTICATED'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask mapIdMgrGroupToRole {-interactive}
```

- Using Jython string:

```
AdminTask.mapIdMgrGroupToRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.mapIdMgrGroupToRole (['-interactive']))
```

removeIdMgrGroupsFromRole

The **removeIdMgrGroupsFromRole** command removes a group from a specified role in federated repositories.

Parameters

-roleName
The name of the role. Valid values are IdMgrAdmin, IdMgrReader, or IdMgrWriter, which are the federated repositories pre-defined roles. (String, required)

-groupId
The common name or unique name of the group to which you want to map the specified role. If you specify the common name, it should correspond to a unique group in the repository. Alternately, to remove the mapping of all logged-in users to the specified role, you can specify a special subject with the value ALLAUTHENTICATED. (String, required)

Note: You can specify an asterisk (*) to remove all users mapped to the specified role.

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl (example 1):

```
$AdminTask removeIdMgrGroupsFromRole {-roleName IdMgrReader
-groupId cn=group1,o=customrealm}
```

Using Jacl (example 2):

```
$AdminTask removeIdMgrGroupsFromRole {-roleName IdMgrReader
-groupId ALLAUTHENTICATED}
```

- Using Jython string (example 1):

```
AdminTask.removeIdMgrGroupsFromRole ('[-roleName IdMgrReader
-groupId cn=group1,o=customrealm]')
```

Using Jython string (example 2):

```
AdminTask.removeIdMgrGroupsFromRole ('[-roleName IdMgrReader
-groupId ALLAUTHENTICATED]')
```

- Using Jython list (example 1):

```
AdminTask.removeIdMgrGroupsFromRole (['-roleName', 'IdMgrReader',
'-groupId', 'cn=group1,o=customrealm'])
```

Using Jython list (example 2):

```
AdminTask.removeIdMgrGroupsFromRole (['-roleName', 'IdMgrReader',
'-groupId', 'ALLAUTHENTICATED'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeIdMgrGroupsFromRole {-interactive}
```

- Using Jython string:

```
AdminTask.removeIdMgrGroupsFromRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeIdMgrGroupsFromRole (['-interactive'])'])
```

removeIdMgrUsersFromRole

The **removeIdMgrUsersFromRole** command removes a user from a specified role in federated repositories.

Parameters

-roleName
The name of the role. Valid values are IdMgrAdmin, IdMgrReader, or IdMgrWriter, which are the federated repositories pre-defined roles. (String, required)

-userId
The user ID or unique name of the user whose mapping to the specified role you want to remove. If you specify the user ID, it should correspond to a unique user in the repository. (String, required)

Note: You can specify an asterisk (*) to remove all users mapped to the specified role.

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeIdMgrUsersFromRole {-roleName IdMgrWriter -userId uid=user1,o=customrealm}
```

- Using Jython string:

```
AdminTask.removeIdMgrUsersFromRole ('[-roleName IdMgrWriter -userId uid=user1,o=customrealm]')
```

- Using Jython list:

```
AdminTask.removeIdMgrUsersFromRole (['-roleName', 'IdMgrWriter',
'-userId', 'uid=user1,o=customrealm'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeIdMgrUsersFromRole {-interactive}
```

- Using Jython string:

```
AdminTask.removeIdMgrUsersFromRole ('[-interactive]'))
```

- Using Jython list:

```
AdminTask.removeIdMgrUsersFromRole (['-interactive'])
```

resetIdMgrConfig

The **resetId MgrConfig** command resets the current configuration to the last configuration that was saved.

Parameters

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Returns

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask resetIdMgrConfig
```

- Using Jython string:

```
AdminTask.resetIdMgrConfig()
```

- Using Jython list:

```
AdminTask.resetIdMgrConfig()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask resetIdMgrConfig {-interactive}
```

- Using Jython string:

```
AdminTask.resetIdMgrConfig ('[-interactive]')
```

- Using Jython list:

```
AdminTask.resetIdMgrConfig (['-interactive'])
```

setIdMgrUseGlobalSchemaForModel

The setIdMgrUseGlobalSchemaForModel command sets the global schema option for the data model in a multiple security domain environment. Global schema refers to the schema of the admin domain.

Note: Application domains that are set to use global schema share the same schema of the admin domain. Hence, if you extend the schema for an application in one domain, you must take into consideration how that might affect applications of other domains as they are also bound by the same schema. For example, adding a mandatory property for one application might cause other applications to fail.

Parameters

-useGlobalSchema
Specifies whether the data model should use the global schema. Global schema refers to the schema of the admin domain. The default value of this parameter is false. (Boolean, required)

-securityDomainName
The name that uniquely identifies the security domain. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrUseGlobalSchemaForModel {-useGlobalSchema true
–securityDomainName mysecDomain}
```

- Using Jython string:

```
AdminTask.setIdMgrUseGlobalSchemaForModel ('[-useGlobalSchema true
–securityDomainName mysecDomain]')
```

- Using Jython list:

```
AdminTask.setIdMgrUseGlobalSchemaForModel (['-useGlobalSchema', 'true',
'–securityDomainName', 'mysecDomain'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrUseGlobalSchemaForModel {-useGlobalSchema true
–securityDomainName mysecDomain}
```

- Using Jython string:

```
AdminTask.setIdMgrUseGlobalSchemaForModel ('[-useGlobalSchema true
–securityDomainName mysecDomain]')
```

- Using Jython list:

```
AdminTask.setIdMgrUseGlobalSchemaForModel (['-useGlobalSchema', 'true',
'–securityDomainName', 'mysecDomain'])
```

showIdMgrConfig

The **showId MgrConfig** command returns the current configuration XML in string format.

Parameters

-file
The name of the file where you want to save the configuration XML string. (String, optional)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Returns

None.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask showIdMgrConfig
```

- Using Jython string:

```
AdminTask.showIdMgrConfig()
```

- Using Jython list:

```
AdminTask.showIdMgrConfig()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask showIdMgrConfig {-interactive}
```

- Using Jython string:

```
AdminTask.showIdMgrConfig ('[-interactive]')
```

- Using Jython list:

```
AdminTask.showIdMgrConfig (['-interactive'])
```

updateIdMgrLDAPBindInfo

The updateIdMgrLDAPBindInfo command dynamically updates the LDAP server bind information. If you specify a value for the **bindDN** parameter, then you must specify a value for the **bindPassword** parameter. If you specify the **id** parameter only, then the LDAP server information is refreshed.

Parameters

-id
The ID of the repository. (String, required)

-bindDN
The binding distinguished name for the LDAP server. (String, optional)

-bindPassword
The binding password for the LDAP server. (String, optional)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jython:

```
AdminTask.updateIdMgrLDAPBindInfo('[-id id1 -bindDN cn=root -bindPassword myPassword22]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPBindInfo(['-id id1 -bindDN cn=root -bindPassword myPassword22'])
```

- Using Jacl:

```
$AdminTask updateIdMgrLDAPBindInfo {-id id1 -bindDN cn=root -bindPassword myPassword22}
```

Interactive mode example usage:

- Using Jython:

```
AdminTask.updateIdMgrLDAPBindInfo(['-interactive'])
```

- Using Jacl:

```
$AdminTask updateIdMgrLDAPBindInfo {-interactive}
```

updateIdMgrSupportedEntityType

The **updateIdMgr Supported EntityType** command updates the configuration that you specify for a supported entity type.

Parameters

-name
The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required)

-securityDomainName
The name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-defaultParent
The default parent node for the supported entity type. (String, optional)

-rdnProperties
The RDN attribute name for the supported entity type in the entity domain name. To reset all the values of the rdnProperties parameter, specify a blank string (""). (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrSupported EntityType {-name entity1}
```

- Using Jython string:

```
AdminTask.updateIdMgrSupported EntityType ('[-name entity1]')
```

- Using Jython list:

```
AdminTask.updateIdMgrSupported EntityType (['-name', 'entity1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrSupported EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrSupported EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrSupported EntityType (['-interactive'])
```

IdMgrRepositoryConfig command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security. The commands and parameters in the IdMgrRepositoryConfig group can be used to create and manage the virtual member manager and LDAP directory properties.

The IdMgrRepositoryConfig command group for the AdminTask object includes the following commands:

addIdMgrLDAPAttr

Use the addIdMgrLDAPAttr command to add an LDAP attribute configuration to the LDAP repository configuration.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

-name
Use this parameter to specify the name of the LDAP attribute used in the repository LDAP adapter. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-propertyName
Use this parameter to specify the name of the corresponding federated repository property. (String, optional)

Note: You cannot add an LDAP attribute configuration for the federated repository properties, principalName and realm.

Note: If you define multiple login properties, then the first login property is programmatically mapped to the federated repositories principalName property. For example, if you set `uid;mail` as the login properties, the LDAP attribute uid value is mapped to the federated repositories principalName property. If you define multiple login properties, after login, the first login property is returned as the value of the principalName property. For example, if you pass `joe@yourco.com` as the principalName value and the login properties are configured as uid;mail, the principalName is returned as joe.

-entityTypes
Use this parameter to specify the entity type which applies the attribute mapping. (String, optional)

-syntax
Use this parameter to specify the syntax of the LDAP attribute. The default value is `string`. For example, the syntax of the `unicodePwd` LDAP attribute is `octetString`. (String, optional)

-defaultValue
Use this parameter to specify the default value of the LDAP attribute. If you do not specify this LDAP attribute when you create an entity which this LDAP attribute applies to, the system adds the attribute using this default value. (String, optional)

-defaultAttr
Use this parameter to specify the default attribute of the LDAP attribute. If you do not specify this LDAP attribute when you create an entity which this LDAP attribute applies to, the system uses this value of the default attribute.

For example, the following configuration defines a samAccountName LDAP attribute with the cn default attribute:

```
<config:attributes name="samAccountName" defaultAttribute="cn">
<config:entityTypes>Group</config:entityTypes>
</config:attributes>
```

In this example, when you create the Group entity, the samAccountName LDAP attribute with the same value as the cn attribute is added to the corresponding LDAP entry.

(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPAttr {-id id_name -name unicode_password -syntax octet_string}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPAttr ('[-id id_name -name unicode_password -syntax octet_string]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPAttr (['-id', 'id_name', '-name', 'unicode_password', '-syntax', 'octet_string'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPAttr {-interactive}
```

- Using Jython:

```
AdminTask.addIdMgrLDAPAttr('-interactive')
```

addIdMgrLDAPAttrNotSupported

Use the addIdMgrLDAPAttrNotSupported command to add a configuration for a federated repository property that the specified LDAP repository does not support.

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

-propertyName
Use this parameter to specify the name of the federated repository property. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-entityTypes
Use this parameter to specify one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPAttrNotSupported {-id id_name -propertyName property_name}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPAttrNotSupported ('[-id id_name –propertyName property_name]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPAttrNotSupported (['-id', 'id_name', '-propertyName', 'property_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPAttrNotSupported {-interactive}
```

- Using Jython:

```
AdminTask.addIdMgrLDAPAttrNotSupported('-interactive')
```

addIdMgrLDAPBackupServer

The **addIdMgrLDAPBackupServer** command sets a backup LDAP server in your configuration.

Required parameters

-id
Specifies the unique ID of the repository. (String, required)

-primary_host
Specifies the primary host of the LDAP server. (String, required)

-host
Specifies the host name for the LDAP server. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-port
Specifies the port number for the LDAP server. (Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPBackupServer {-id id_name -primary_host host_name1 -host host_name2 -port port_number}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPBackupServer ('[-id id_name -primary_host host_name1 -host host_name2 -port port_number]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPBackupServer (['-id', 'id_name', '-primary_host', 'host_name1', '-host', 'host_name2', '-port', 'port_number'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPBackupServer {-interactive}
```

- Using Jython:

```
AdminTask.addIdMgrLDAPBackupServer('-interactive')
```

addIdMgrLDAPEntityType

The **addIdMgrLDAPEntityType** command adds an LDAP entity type definition.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the entity type. (String, required)

-objectClasses
One or more object classes for the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-searchFilter
The search filter that you want to use to search the entity type. (String, optional)

-objectClassesForCreate
The object class to use when an entity type is created. If the value of this parameter is the same as the objectClass parameter, you do not need to specify this parameter. (String, optional)

-searchBases
The search base or bases to use while searching the entity type. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPEntity Type {-id id_name -name name_value -objectClasses object_class}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPEntity Type ('[-id id_name -name name_value -objectClasses object_class]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPEntity Type (['-id', 'id_name', '-name', 'name_value', '-objectClasses', 'object_class'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAP EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAP EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAP EntityType (['-interactive'])
```

addIdMgrLDAPEntityTypeRDNAttr

The **addId MgrLDAP EntityType RDNAttr** command adds RDN attribute configuration to an LDAP entity type definition.

Required parameters

-id
The ID of the repository. (String, required)

-entityTypeName
The name of the entity type. (String, required)

-name
The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-objectClass
The object class to use for the entity type for the relative distinguished name (RDN) attribute name that you specify. Use this parameter to map one entity type to multiple structural object classes. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPEntity TypeRDNAttr {-id id_name -entityTypeName entity_type -name name_value}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPEntity TypeRDNAttr ('[-id id_name -entityTypeName entity_type -name name_value]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPEntity TypeRDNAttr (['-id', 'id_name', '-entityTypeName', 'entity_type', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPEntity TypeRDNAttr {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPEntity TypeRDNAttr (['-interactive'])
```

addIdMgrLDAPExternalIdAttr

Use the addIdMgrLDAPExternalIdAttr command to add a configuration for an LDAP attribute that is used as an external ID in the specified LDAP repository.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

-name
Use this parameter to specify the name of the external ID attribute of the LDAP repository. (String, required)

Important: Specify `distinguishedName` as the value of this parameter to indicate that the distinguished name (DN) of the entity is used as the external ID.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-entityTypes
Use this parameter to specify one or more entity types. Use a semicolon (;) as the delimiter to specify multiple entity types. (String, optional)

-syntax
Use this parameter to specify the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString. (String, optional)

-wimGenerate
Use this parameter to indicate whether the federated repository generates the value of the LDAP attribute. The default value is false. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPExternalIdAttr {-id id_name -name unicodePwd -syntax octetString}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPExternalIdAttr ('[-id id_name -name unicode_password -syntax octet_string]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPExternalIdAttr (['-id', 'id_name', '-name', 'unicode_password', '-syntax', 'octet_string'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPExternalIdAttr {-interactive}
```

- Using Jython:

```
AdminTask.addIdMgrLDAPExternalIdAttr('-interactive')
```

addIdMgrLDAPGroupDynamicMemberAttr

The **addIdMgr LDAPGroup Dynamic Member Attr** command adds a dynamic member attribute configuration to an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required)

-objectClass
The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUniqueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-id id_name -name name_value -objectClass object_class}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-id id_name -name name_value -objectClass object_class]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-id', 'id_name', '-name', 'name_value', '-objectClass', 'object_class'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])
```

addIdMgrLDAPGroupMemberAttr

The **addIdMgr LDAPGroup MemberAttr** command adds a member attribute configuration to an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required)

-scope
The scope of the member attribute. The valid values for this parameter include the following:

- direct - The member attribute only contains direct members, therefore, this value refers to the member directly contained by the group and not contained through the nested group. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct member of Group1 but User1 is not a direct member of Group1. Both `member` and `uniqueMember` are direct member attributes.
- nested - The member attribute that contains the direct members and the nested members.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-dummyMember
Indicates that if you create a group without specifying a member, a dummy member will be filled in to avoid creating an exception about missing a mandatory attribute. (String, optional)

-objectClass
The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUniqueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPGroup MemberAttr {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPGroup MemberAttr ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPGroup MemberAttr (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPGroup MemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPGroup MemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPGroup MemberAttr (['-interactive'])
```

addIdMgrLDAPServer

The **addId MgrLDAP Server** command adds an LDAP server to the LDAP repository ID that you specify.

Required parameters

-id
The ID of the repository. (String, required)

-host
The host name for the primary LDAP server. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-port
The port number for the LDAP server. (Integer, optional)

-bindDN
The binding distinguished name for the LDAP server. (String, optional)

-bindPassword
The binding password. (String, optional)

-authentication
Indicates the authentication method to use. The default value is `simple`. Valid values include: `none` or `strong`. (String, optional)

-referal
The LDAP referral. The default value is `ignore`. Valid values include: `follow`, `throw`, or `false`. (String, optional)

-derefAliases
Controls how aliases are dereferenced. The default value is `always`. Valid values include:

- `never` - never deference aliases
- `finding` - deferences aliases only during name resolution
- `searching` - deferences aliases only after name resolution

(String, optional)

-sslEnabled
Indicates to enable SSL or not. The default value is `false`. (Boolean, optional)

-connectionPool
The connection pool. The default value is `false`. (Boolean, optional)

-connectTimeout
The connection timeout in seconds. The default value is `20`. (Integer, optional)

Restriction: Due to a current JNDI limitation, the maximum connection timeout is 20 seconds. Even if you specify a value above 20 seconds, the connection still times out at 20 seconds.

-ldapServerType
The type of LDAP server being used. The default value is `IDS51`. (String, optional)

-sslConfiguration
The SSL configuration. (String, optional)

-certificateMapMode
Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is EXACT_DN. To use the certificate filter for the mapping, specify FILTERDESCRIPTORMODE. (String, optional)

-certificateFilter
If certificateMapMode has the value FILTERDESCRIPTORMODE, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. For more information, see the section Certificate filter in the topic, Lightweight Directory Access Protocol repository configuration settings. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAPServer {-id id_name -host myhost.ibm.com}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAPServer ('[-id id_name -host myhost.ibm.com]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAPServer (['-id', 'id_name', '-host', 'myhost.ibm.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrLDAP Server {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrLDAP Server ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrLDAP Server (['-interactive'])
```

addIdMgrRepositoryBaseEntry

The **addIdMgr Repository BaseEntry** command adds a base entry to the specified repository.

Required parameters

-id
The ID of the repository. (String, required)

-name
The distinguished name of a base entry. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-nameInRepository
The distinguished name in the repository that uniquely identifies the base entry name. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrRepositoryBaseEntry {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.addIdMgrRepositoryBaseEntry ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.addIdMgrRepositoryBaseEntry (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrRepositoryBaseEntry {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrRepositoryBaseEntry ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrRepositoryBaseEntry (['-interactive'])
```

createIdMgrCustomRepository

The **createIdMgrCustomRepository** command creates a custom repository configuration.

Required parameters

-id
The ID of the repository. (String, required)

-adapterClassName
The implementation class name for the repository adapter. (String, required)

Examples

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrCustomRepository {-id id_name -adapterClassName adapter_class_name}
```

- Using Jython string:

```
AdminTask.createIdMgrCustomRepository('-id id_name -adapterClassName adapter_class_name')
```

- Using Jython list:

```
AdminTask.createIdMgrCustomRepository(['-id', 'id_name', '-adapterClassName', 'adapter_class_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrCustomRepository {-interactive}
```

- Using Jython:

```
AdminTask.createIdMgrCustomRepository('-interactive')
```

createIdMgrDBRepository

The **createIdMgrDBRepository** command creates a database repository configuration.

Required parameters

-id
The ID of the repository. (String, required)

-dataSourceName
The name of the data source. The default value is `jdbc/wimDS`. (String, required)

-databaseType
The type of the database. The default value is `DB2`. (String, required)

-dbURL
The URL of the database. (String, required)

-dbAdminId
The database administrator ID. (String, required if database type is not Apache Derby.)

-dbAdminPassword
The database administrator password. (String, required if database type is not Apache Derby.)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-adapterClassName
The default value is `com.ibm.ws.wim.adapter.db.DBAdapter`. (String, optional)

-JDBCDriverClass
The JDBC driver class name. (String, optional)

-supportSorting
Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional)

-supportTransactions
Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional)

-isExtIdUnique
Specifies if the external ID is unique. The default value is `true`. (Boolean, optional)

-supportExternalName
Indicates if external names are supported or not. The default value is `false`. (Boolean, optional)

-supportAsyncMode
Indicates if the adapter supports async mode or not. The default value is false. (Boolean, optional)

-readOnly
Indicates if this is a read only repository. The default value is false. (Boolean, optional)

-entityRetrievalLimit
Indicates the value of the retrieval limit on database entries. The default value is `200`. (Integer, optional)

-saltLength
The salt length in bits. The default value is `12`. (Integer, optional)

-encryptionKey
The default value is `rZ15ws0ely9yHk3zCs3sTMv/ho8fY17s`. (String, optional)

-dbSchema
The database schema of the database repository that you want to configure. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user. (String, optional).

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrDB Repository {-id id_name -dataSourceName datasource_name -databaseType database_type}
```

- Using Jython string:

```
AdminTask.createIdMgrDB Repository ('[-id id_name -dataSourceName datasource_name -databaseType database_type]')
```

- Using Jython list:

```
AdminTask.createIdMgrDB Repository (['-id', 'id_name', '-dataSourceName', 'datasource_name', '-databaseType', 'database_type'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrDB Repository {-interactive}
```

- Using Jython string:

```
AdminTask.createIdMgrDB Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createIdMgrDB Repository (['-interactive'])
```

createIdMgrFileRepository

The **createId MgrFile Repository** command creates a file repository configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-messageDigest Algorithm
The message digest algorithm that will be used for hashing the password. The default value is `SHA-1`. Valid values include the following: `SHA-1`, `SHA-384`, or `SHA-512`.(String, optional)

-adapterClassName
The default value is `com.ibm.ws. wim.adapter. file.was. FileAdapter`. (String, optional)

-supportPaging
Indicates if paging is supported or not. The default value is `false`. (Boolean, optional)

-supportSorting
Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional)

-supportTransactions
Indicates if transaction is supported or not. The default value is `false`. (Boolean, optional)

-isExtIdUnique
Specifies if the external ID is unique or not. The default value is `true`. (Boolean, optional)

-supportAsyncMode
Indicates if the adapter supports async mode or not. The default value is false. (Boolean, optional)

-supportExternalName
Indicates if external names are supported or not. The default value is `false`. (Boolean, optional)

-baseDirectory
The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional)

-fileName
The file name of the repository. The default value is `fileRegistry.xml`. (String, optional)

-saltLength
The salt length of the randomly generated salt for password hashing. The default value is `12`. (Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrFile Repository {-id id_name -messageDigestAlgorithm algorithm_value}
```

- Using Jython string:

```
AdminTask.createIdMgrFile Repository ('[-id id_name -messageDigestAlgorithm algorithm_value]')
```

- Using Jython list:

```
AdminTask.createIdMgrFile Repository (['-id', 'id_name', '-messageDigestAlgorithm', 'algorithm_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrFile Repository {-interactive}
```

- Using Jython string:

```
AdminTask.createIdMgrFile Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createIdMgrFile Repository (['-interactive'])
```

createIdMgrLDAPRepository

The **create IdMgrLDAP Repository** command creates an LDAP repository configuration.

Required parameters

-id
The unique identifier for the repository. (String, required)

-ldapServerType
The type of LDAP server that is being used. The default value is `IDS51`. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-adapterClassName
The default value is `com.ibm.ws. wim.adapter. ldap.LdapAdapter`. (String, optional)

-supportSorting
Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional)

-supportPaging
Indicates if paging is supported or not. The default value is `false`. (Boolean, optional)

-supportTransactions
Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional)

-isExtIdUnique
Specifies if the external ID is unique. The default value is `true`. (Boolean, optional)

-supportAsyncMode
Indicates if the adapter supports async mode or not. The default value is false. (Boolean, optional)

-supportExternalName
Indicates if external names are supported or not. The default value is `false`. (Boolean, optional)

-certificateMapMode
Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify the value as `certificatefilter`. (String, optional)

-certificateFilter
If the certificateMapMode parameter has the value `certificatefilter`, then this property specifies the LDAP filter that maps attributes in the client certificate to entries in LDAP. (String, optional)

-loginProperties
Indicates the property name used for login. (String , optional)

Note: If you define multiple login properties, then the first login property is programmatically mapped to the federated repositories principalName property. For example, if you set `uid;mail` as the login properties, the LDAP attribute uid value is mapped to the federated repositories principalName property. If you define multiple login properties, after login, the first login property

is returned as the value of the principalName property. For example, if you pass joe@yourco.com as the principalName value and the login properties are configured as uid;mail, the principalName is returned as joe.

-sslConfiguration
The SSL configuration. (String, optional)

-translateRDN
Indicates to translate RDN or not. The default value is false. (Boolean, optional)

-searchTimeLimit
The value of search time limit. (Integer, optional)

-searchCountLimit
The value of search count limit. (Integer, optional)

-searchPageSize
The value of search page size. (Integer, optional)

-returnToPrimaryServer
(Integer, optional)

-primaryServerQueryTimeInterval
(Integer, optional)

-default
If you set this parameter to true, the default values will be set for the remaining configuration properties of the LDAP repository. (Boolean, optional)

-supportChangeLog
This parameter indicates whether the repository supports change tracking. Valid values for this parameter are none or native. The default value is none. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrLDAP Repository {-id id_name -ldapServerType LDAP_server_type}
```

- Using Jython string:

```
AdminTask.createIdMgrLDAP Repository ('[-id id_name -ldapServerType LDAP_server_type]')
```

- Using Jython list:

```
AdminTask.createIdMgrLDAP Repository (['-id', 'id_name', '-ldapServerType', 'LDAP_server_type'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrLDAP Repository {-interactive}
```

- Using Jython string:

```
AdminTask.createIdMgrLDAP Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createIdMgrLDAP Repository (['-interactive'])
```

deleteIdMgrLDAPAttr

Use the deleteIdMgrLDAPAttr command to delete the LDAP attribute configuration data for a specific entity type from the LDAP repository of interest.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

Note: The deleteIdMgrLDAPAttr command also requires the name of either the LDAP attribute or federated repository property. Specify a value for either the **-name** or **-propertyName** parameter that is described in the next section. However, do not specify both parameters. Although the **-name** or **-propertyName** parameters are designated as optional parameters, an error occurs if you do not specify one of the parameters or if you specify both parameters.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-name
Use this parameter to specify the name of the LDAP attribute used in the repository LDAP adapter. (String, required)

-entityTypes
Use this parameter to specify the entity type which applies the attribute mapping. (String, optional)

-propertyName
Use this parameter to specify the name of the corresponding federated repository property. (String optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPAttr {-id id_name -name unicode_password}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPAttr ('[-id id_name -name unicode_password]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPAttr (['-id', 'id_name', '-name', 'unicode_password'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPAttr {-interactive}
```

- Using Jython:

```
AdminTask.deleteIdMgrLDAPAttr('-interactive')
```

deleteIdMgrLDAPAttrNotSupported

Use the deleteIdMgrLDAPAttrNotSupported command to delete the configuration for a federated repository property that the specified LDAP repository does not support.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

-propertyName
Use this parameter to specify the name of the federated repository property. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-entityTypes
Use this parameter to specify one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types. If you do not specify this parameter, the deleteIdMgrLDAPAttrNotSupported command deletes all the configuration data of the specified attribute. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPAttrNotSupported {-id id_name -propertyName property_name}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPAttrNotSupported ('[-id id_name –propertyName property_name]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPAttrNotSupported (['-id', 'id_name', '-propertyName', 'property_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPAttrNotSupported {-interactive}
```

- Using Jython:

```
AdminTask.deleteIdMgrLDAPAttrNotSupported('-interactive')
```

deleteIdMgrLDAPEntityType

The **deleteId MgrLDAP EntityType** command deletes the LDAP entity type configuration data for a specified entity type for a specific LDAP repository.

Parameters and return values

-id
The ID of the repository. (String, required)

-name
The name of the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP EntityType {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP EntityType ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP EntityType (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP EntityType (['-interactive'])
```

deleteIdMgrLDAPEntityTypeRDNAttr

The **deleteId MgrLDAP EntityType RDNAttr** command deletes the relative distinguished name (RDN) attribute configuration from an LDAP entity type configuration.

Required parameters

-id
The ID of the repository. (String, required)

-entityTypeName
The name of the entity type. (String, required)

-name
The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPEntity TypeRDNAttr {-id id_name -name name_value -entityTypeName entity_type}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP EntityTypeRDNAttr ('[-id id_name -name name_value -entityType Name entity_type]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr (['-id', 'id_name', '-name', 'name_value', '-entity TypeName', 'entity_type'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPEntity TypeRDNAttr {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr (['-interactive'])
```

deleteIdMgrLDAPExternalIdAttr

Use the deleteIdMgrLDAPExternalIdAttr command to delete the configuration for an LDAP attribute that is used as an external ID in the specified LDAP repository.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

-name
Use this parameter to specify the name of the external ID attribute of the LDAP repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-entityTypes
Use this parameter to specify one or more entity types. Use a semicolon (;) as the delimiter to specify multiple entity types. If you do not specify this parameter, the deleteIdMgrLDAPExternalIdAttr command deletes all the configuration data of the specified attribute. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPExternalIdAttr {-id id_name -name unicode_password}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPExternalIdAttr ('[-id id_name -name unicode_password]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPExternalIdAttr (['-id', 'id_name', '-name', 'unicode_password'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPExternalIdAttr {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPExternalIdAttr ('-interactive')
```

deleteIdMgrLDAPGroupConfig

The **deleteId MgrLDAP GroupConfig** command deletes the LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP GroupConfig {-id id_name}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP GroupConfig ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP GroupConfig (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP GroupConfig {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP GroupConfig ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP GroupConfig (['-interactive'])
```

deleteIdMgrLDAPGroupMemberAttr

The **deleteIdMgr LDAPGroup MemberAttr** command deletes a member attribute configuration from an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute, for example, member or uniqueMember. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPGroupMemberAttr {-id id_name -name attr_name}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPGroupMemberAttr (['-id', 'id_name', '-name', 'attr_name'])
```

- Using Jython list:

```
$AdminTask deleteIdMgrLDAPGroupMemberAttr {-id id_name -name attr_name}
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP GroupMemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP GroupMemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP GroupMemberAttr (['-interactive'])
```

deleteIdMgrLDAPGroupDynamicMemberAttr

The **deleteIdMgr LDAPGroup Dynamic MemberAttr** command deletes a dynamic member attribute configuration from an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute. For example, `memberURL`. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP GroupDynamicMemberAttr {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP GroupDynamicMemberAttr ('[ -id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP GroupDynamicMemberAttr ([' -id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAPGroup DynamicMemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])
```

deleteIdMgrLDAPServer

The **deleteId MgrLDAP Server** command deletes the configuration for the LDAP server that you specify from the LDAP repository ID that you specify.

Required parameters

-id
The ID of the repository. (String, required)

-host
The host name for the primary LDAP server. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP Server {-id id_name -host myhost.ibm.com}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP Server ('[-id id_name -host myhost.ibm.com]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP Server (['-id', 'id_name', '-host', 'myhost.ibm.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrLDAP Server {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrLDAP Server ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrLDAP Server (['-interactive'])
```

deleteIdMgrRepository

The **deleteIdMgr Repository** command deletes a repository that you specify.

Required parameters

-id
The ID of the repository. Valid values include existing repository IDs. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgr Repository {-id id_name}
```

- Using Jython string:

```
AdminTask.deleteIdMgr Repository ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.deleteIdMgr Repository (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRepos itory {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrReposi tory ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrReposi tory (['-interactive'])
```

deleteIdMgrRepositoryBaseEntry

The **deleteIdMgr Repository BaseEntry** command deletes a base entry from the specified repository.

Required parameters

-id
The ID of the repository. (String, required)

-name
The distinguished name of a base entry. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRepository BaseEntry {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRepository BaseEntry ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRepository BaseEntry (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRepository BaseEntry {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRepository BaseEntry ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRepository BaseEntry (['-interactive'])
```

getIdMgrLDAPAttrCache

The **getIdMgr LDAPAttr Cache** command returns the LDAP attribute cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPAttr Cache {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPAttr Cache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPAttr Cache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAP AttrCache {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPAttr Cache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPAttr Cache (['-interactive'])
```

getIdMgrLDAPContextPool

The **getIdMgr LDAP Context Pool** command returns the LDAP context pool configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPContext Pool {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPContext Pool ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPContext Pool (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPCon textPool {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPCon textPool ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPCon textPool (['-interactive'])
```

getIdMgrLDAPEntityType

The **getIdMgr LDAP EntityType** command returns the LDAP entity type configuration data.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPEntity Type {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPEntity Type ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPEntity Type (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPEn tityType {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPEn tityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPEn tityType (['-interactive'])
```

getIdMgrLDAPEntityTypeRDNAttr

The **getIdMgr LDAPEntity TypeRDNAttr** command returns the relative distinguished name (RDN) attribute configuration for an LDAP entity type definition.

Required parameters

-id
The ID of the repository. (String, required)

-entityTypeName
The name of the entity name. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-id id_name -entityTypeName name_value}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-id id_name -entityTypeName name_value]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-id', 'id_name', '-entityTypeName', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-interactive'])
```

getIdMgrLDAPGroupConfig

The **getIdMgr LDAPG roupConfig** command returns the LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPGroup Config {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPGroup Config ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPGroup Config (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPGroup Config {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPGroup Config ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPGroup Config (['-interactive'])
```

getIdMgrLDAPGroupDynamicMemberAttrs

The **getIdMgr LDAPGroup Dynamic Member Attrs** command returns the dynamic member attribute configuration from the LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPGroupDynamic MemberAttrs {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPGroupDynamic MemberAttrs ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPGroupDynamic MemberAttrs (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPGroup DynamicMemberAttrs {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPGroup DynamicMemberAttrs ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPGroup DynamicMemberAttrs (['-interactive'])
```

getIdMgrLDAPGroupMemberAttrs

The **getIdMgr LDAPGroup MemberAttrs** command returns the member attribute configuration for the LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPGroup MemberAttrs {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPGroup MemberAttrs ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPGroup MemberAttrs (['-interactive'])
```

getIdMgrLDAPSearchResultCache

The **getIdMgr LDAPSearch ResultCache** command returns the LDAP search result cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAP SearchResultCache {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAP SearchResultCache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPSear chResultCache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPSea rchResultCache {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPSear chResultCache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPSe archResultCache (['-interactive'])
```

getIdMgrLDAPServer

The **getIdMgr LDAPServer** command returns the configuration for the LDAP server that you specify for the LDAP repository ID that you specify.

Required parameters

-id
The ID of the repository. (String, required)

-host
The host name for the primary LDAP server. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPServer {-id id_name -host myhost.ibm.com}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPServer ('[-id id_name -host myhost.ibm.com]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPServer (['-id', 'id_name', '-host', 'myhost.ibm.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrLDAPServer {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrLDAPServer ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrLDAPServer (['-interactive'])
```

getIdMgrRepository

The **getIdMgr Repository** command returns the configuration of the specified repository.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRepos itory {-id id_name}
```

- Using Jython string:

```
AdminTask.getIdMgrReposi tory ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.getIdMgrReposi tory (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRepos itory {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrReposi tory ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrReposi tory (['-interactive'])
```

listIdMgrLDAPAttrs

Use the listIdMgrLDAPAttrs command to list the name of each configured attributes for the LDAP repository of interest.

Required parameters

-id
Use thois parameter to specify the unique ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Return value

The command returns a list of HashMaps that contains parameters of the addIdMgrLDAPAttr command as keys. For the **entityTypes** parameter, which is multivalued, the value of the key is a string that is delimited by a semicolon (;). The return value includes an additional key called entityTypesList. The value of the entityTypesList key is a List object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPAttrs {-id id_value}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAPAttrs ('[-id id_value]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAPAttrs (['-id', 'id_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPAttrs {-interactive}
```

- Using Jython:

```
AdminTask.listIdMgrLDAPAttrs('-interactive')
```

listIdMgrLDAPAttrsNotSupported

Use the listIdMgrLDAPAttrsNotSupported command to list the details of all configured federated repository properties that the specified LDAP repository does not support.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Return value

The command returns a List of HashMaps that contains parameters of the addIdMgrLDAPAttrNotSupported command as keys. For multivalued parameters such as **entityTypes**, the value of the key is a List object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPAttrsNotSupported {-id id_name}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAPAttrsNotSupported ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAPAttrsNotSupported (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPAttrsNotSupported ('[-interactive]')
```

- Using Jython:

```
AdminTask.listIdMgrLDAPAttrsNotSupported ('-interactive')
```

listIdMgrCustomProperties

The **listIdMgr Custom Properties** command returns a list of custom properties for the repository that you specify.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrCustomPro perties {-id id_value}
```

- Using Jython string:

```
AdminTask.listIdMgrCustom Properties ('[-id id_value]')
```

- Using Jython list:

```
AdminTask.listIdMgrCustom Properties (['-id', 'id_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrCustom Properties {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrCustom Properties ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrCustom Properties (['-interactive'])
```

listIdMgrLDAPBackupServers

The **listIdMgr LDAPBack upServers**command returns a list of the backup LDAP server or servers.

Required parameters and return values

-id
The ID of the repository. (String, required)

-primary_host
The host name for the primary LDAP server. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP BackupServer {-id id_value -primary_host host_name}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAPBack upServer ('[-id id_value -primary_host host_name]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAPBack upServer (['-id', 'id_value', '-primary_host', 'host_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP BackupServer {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAP BackupServer ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAP BackupServer (['-interactive'])
```

listIdMgrLDAPEntityTypes

The **listIdMgr LDAPEn tityTypes** command lists the name of all of the configured LDAP entity type definitions.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP EntityType {-id id_value}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAP EntityType ('[-id id_value]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAPEntity Type (['-id', 'id_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAP EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAP EntityType (['-interactive'])
```

listIdMgrLDAPExternalIdAttrs

Use the listIdMgrLDAPExternalIdAttrs command to list the details of all LDAP attributes used as an external ID in the specified LDAP repository.

Target object

None

Required parameters

-id
Use this parameter to specify the unique ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Return value

The command returns a List of HashMaps that contains parameters of the addIdMgrLDAPExternalIdAttr command as keys. For multivalued parameters such as **entityTypes**, the value of the key is a List object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPExternalIdAttrs {-id id_name}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAPExternalIdAttrs ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAPExternalIdAttrs (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAPExternalIdAttrs ('[-interactive]')
```

- Using Jython string:

```
AdminTask.listIdMgrLDAPExternalIdAttrs('-interactive')
```

listIdMgrLDAPServers

The **listIdMgr LDAP Servers** command lists all of the configured primary LDAP servers.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP Servers {-id id_value}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAP Servers ('[-id id_value]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAP Servers (['-id', 'id_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrLDAP Servers {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrLDAP Servers ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrLDAP Servers (['-interactive'])
```

listIdMgrRepositories

The **listIdMgr Repositories** command lists names and types of all configured repositories.

Required parameters and return values

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

- Returns: A hash map with key as the name of the repository and value as another hash map that includes the following keys:
  - repositoryType - The type of repository. For example, File, LDAP, DB, and so on.
  - specificRepositoryType - The specific type of repository. For example, LDAP, IDS51, NDS, and so on.
  - host - The host name where the repository resides. For File, it is LocalHost and for DB it is dataSourceName.

  This command will not return the Property Extension and Entry Mapping repository data.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRepositories
```

- Using Jython string:

```
AdminTask.listIdMgrRepositories()
```

- Using Jython list:

```
AdminTask.listIdMgrRepositories()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrReposi tories {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrReposit ories ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrReposi tories (['-interactive'])
```

listIdMgrRepositoryBaseEntries

The **listIdMgr Repository BaseEntries** command lists the base entries for a specified repository.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRepository BaseEntries {-id id_value}
```

- Using Jython string:

```
AdminTask.listIdMgrRepository BaseEntries ('[-id id_value]')
```

- Using Jython list:

```
AdminTask.listIdMgrRepository BaseEntries (['-id', 'id_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRepository BaseEntries {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrRepository BaseEntries ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrRepository BaseEntries (['-interactive'])
```

listIdMgrSupportedDBTypes

The **listIdMgr Supported DBTypes** command returns a list of supported database types.

Required parameters

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupportedDBTypes
```

- Using Jython string:

```
AdminTask.listIdMgrSupportedDBTypes()
```

- Using Jython list:

```
AdminTask.listIdMgrSupportedDBTypes()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupported DBTypes {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrSupported DBTypes ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrSupported DBTypes (['-interactive'])
```

listIdMgrSupportedMessageDigestAlgorithms

The **listIdMgr Supported Message Digest Algorithms** command returns a list of supported message digest algorithms.

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupported MessageDigestAlgorithms
```

- Using Jython string:

```
AdminTask.listIdMgrSupported MessageDigestAlgorithms()
```

- Using Jython list:

```
AdminTask.listIdMgrSupported MessageDigestAlgorithms()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupportedMes sageDigestAlgorithms {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrSupportedMessage DigestAlgorithms ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrSupportedMessage DigestAlgorithms (['-interactive'])
```

listIdMgrSupportedLDAPServerTypes

The **listIdMgr Supported LDAP ServerTypes** command returns a list of supported LDAP server types.

Required parameters

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupported LDAPServerTypes
```

- Using Jython string:

```
AdminTask.listIdMgrSupported LDAPServerTypes()
```

- Using Jython list:

```
AdminTask.listIdMgrSupported LDAPServerTypes()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrSupported LDAPServerTypes {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrSupported LDAPServerTypes ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrSupported LDAPServerTypes (['-interactive'])
```

removeIdMgrLDAPBackupServer

The **removeIdMgr LDAPBack upServer**command removes the backup LDAP server or servers.

Required parameters

-id
The ID of the repository. (String, required)

-primary_host
The host name for the primary LDAP server. (String, required)

-host
The name of the backup host name. Use a asterisk (*) if you want to remove all backup servers. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-port
The port number of the LDAP server. (Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeIdMgrLDAP BackupServer {-id id_value -primary_host myprimaryhost.ibm.com -host myhost.ibm.com}
```

- Using Jython string:

```
AdminTask.removeIdMgrLDAPBackup Server ('[-id id_value -primary_host myprimaryhost.ibm.com -host myhost.ibm.com]')
```

- Using Jython list:

```
AdminTask.removeIdMgrLDAPBack upServer (['-id', 'id_value', '-primary_host', 'myprimaryhost.ibm.com', '-host', 'myhost.ibm.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeIdMgrLDAP BackupServer {-interactive}
```

- Using Jython string:

```
AdminTask.removeIdMgrLDAP BackupServer ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeIdMgrLDAP BackupServer (['-interactive'])
```

setIdMgrCustomProperty

The **setIdMgr Custom Property** command : sets, adds or deletes a custom property to a repository configuration. If a value is not specified, or if there is an empty string, the property is deleted from the repository configuration. If a name does not exist it is added if a value is specified. If the name is "*" then all of the custom properties are deleted.

Required parameters

-id
The unique identifier of the repository. Valid values include the existing repository IDs. (String, required)

-name
The name of the additional property for the repository that are not defined OOTB.(String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-value
The value of a property for the repository. If this parameter is an empty string, the property is deleted from the repository configuration. If this parameter is not an empty string, and a name does not exist, it is added. If a name is an empty string, all of the custom properties are deleted. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrCustomProperty {-id id_value -name name_value -value value}
```

- Using Jython string:

```
AdminTask.setIdMgrCustomProperty ('[-id id_name -name name_value -value value]')
```

- Using Jython list:

```
AdminTask.setIdMgrCustomProperty (['-id', 'id_name', '-name', 'name_value', '-value', 'value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrCustom Property {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrCustom Property ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrCustom Property (['-interactive'])
```

setIdMgrLDAPAttrCache

The **setIdMgr LDAPA ttrCache** command configures the LDAP attribute cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cachesDiskOffLoad
(String, optional)

-enabled
Indicates if you want to enable attribute caching. The default value is `true`. (Boolean, optional)

-cacheSize
The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is `100`. The default value is `4000`. (Integer, optional)

-cacheTimeOut
The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is `0`. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is `1200`. (Integer, optional)

-attributeSizeLimit
An integer that represents the maximum number of attribute object values that can cache in the attributes cache.

Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is `2000`. (Integer, optional)

-serverTTLAttribute
The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out.

The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of `0` implies that the object will not be cached. For more information about this attribute, go to: http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt.

The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttyl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.

For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional)

-cacheDistPolicy
The distribution policy for the dynamic cache in a cluster environment.

The valid values are `none` (for NOT_SHARED), `push` (for SHARED_PUSH), and `push_pull` (for SHARED_PUSH_PULL) and the default value is `none`. The value of this parameter is read during the adapter startup process and the cache policy is set accordingly.(String, optional)

- Returns: None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPAttr Cache {-id id_name}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPAttr Cache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPAttr Cache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPAttr Cache {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPAttr Cache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPAttr Cache (['-interactive'])
```

setIdMgrLDAPContextPool

The **setIdMgr LDAPCo ntextPool** command sets up the LDAP context pool configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-enabled
By default, the context pool is enabled. If you set this parameter to `false`, the context pool is disabled. When the context pool is disabled, new context instances will be created for each request. The default value is `true`. (Boolean, optional)

-initPoolSize
The number of context instances that the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is `1` to `50`. The default value is `1`. (Integer, optional)

-maxPoolSize
The maximum number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When the pool size reaches this number, new context instances cannot be created for new requests. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the amount of time that you specify using the poolWaitTime parameter.

The minimum value for this parameter is `0`. There is no maximum value. Setting the value of this parameter to `0` means that there is no maximum size and a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is `0`.(Integer, optional)

-prefPoolSize
The preferred number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, the context pool creates and uses a new pooled context instance regardless of whether an idle connection is available. When a request finishes with a pooled context instance and the pool size is greater than the preferred size, the context pool closes and removes the pooled context instance from the pool.

The valid range for this parameter is from 0 to 100. Setting the value of this parameter to 0 means that there is no preferred size and a request for a pooled context instance results in a newly created context instance only if no idle ones are available. The default value is 3.(Integer, optional)

-poolTimeOut
An integer that represents the number of seconds that an idle context instance might remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection is closed no matter whether this context instance is stale or active. A new context instance is created and put back to the pool after it has been released from the request.

The minimum value for this parameter is 0. There is no maximum value. Setting the value of this parameter to 0 means that the context instances in the pool remain until they are staled. The context pool catches the communication exception and recreates a new context instance. The default value is 0.(Integer, optional)

-poolWaitTime
The time interval in milliseconds that the request waits until the context pool rechecks if there are idle context instances available in the pool when the number of context instances reaches the maximum pool size. If no idle context instance, the request will continue waiting for the same period of time until next checking.

The minimum value for the poolWaitout parameter is 0. There is no maximum value. A value of 0 for this parameter means that the context pool will not check if idle context exists. The request will be notified when a context instance releases from other requests. The default value is 3000.(Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPCon textPool {-id id_name}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPCon textPool ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPCon textPool (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPCon textPool {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPCon textPool ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPCon textPool (['-interactive'])
```

setIdMgrLDAPGroupConfig

The **setIdMgr LDAPGr oupConfig** command sets up the LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-updateGroup Membership
Updates the group membership if the member is deleted or renamed. Some LDAP servers, for example, Domino server, do not clean up the membership of the user when a user is deleted or renamed. If you choose these LDAP server types in the ldapServerType property, the value of this parameter is set to `true`. Use this parameter to change the value. The default value is `false`. (Boolean, optional)

-name
The name of the membership attribute. For example, `memberOf` in an active directory server and `ibm-allGroups` in IDS. (String, optional)

-scope
The scope of the membership attribute. The following are the possible values for this parameter:

- direct - The membership attribute only contains direct groups. Direct groups contain the member and are not contained through a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct group of user1, but group1 is not a direct group of user1.
- nested - The membership attribute contains both direct groups and nested groups.
- all - The membership attribute contains direct groups, nested groups, and dynamic members.

The default value is `direct`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAP GroupConfig {-id id_name}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAP GroupConfig ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAP GroupConfig (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPGr oupConfig {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPGr oupConfig ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPGroup Config (['-interactive'])
```

setIdMgrLDAPSearchResultCache

The **setIdMgr LDAPSearch ResultCache** command sets up the LDAP search result cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cachesDiskOffLoad
Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is `false`. (Boolean, optional)

-enabled
Enables the search results cache. The default value is `true`. (Boolean, optional)

-cacheSize
The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is `100`. The default value is `2000`. (Integer, optional)

-cacheTimeOut
The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is `0`. A value of `0` means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is `600`. (Integer, optional)

-searchResultSizeLimit
The maximum number of entries contained in the naming enumeration object that can be cached in the search results cache.For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to `1000`. The default value is `1000`. (Integer, optional)

-cacheDistPolicy
The distribution policy for the dynamic cache in a cluster environment.

The valid values are `none` (for NOT_SHARED), `push` (for SHARED_PUSH), and `push_pull` (for SHARED_PUSH_PULL) and the default value is `none`. The value of this parameter is read during the adapter startup process and the cache policy is set accordingly.(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPSearch ResultCache {-id id_name}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPSearch ResultCache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPSearch ResultCache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrLDAPSearch ResultCache {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrLDAPSearch ResultCache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrLDAPSearch ResultCache (['-interactive'])
```

setIdMgrEntryMappingRepository

The **setIdMgr Entry Mapping Repository** command sets or updates an entry mapping repository configuration.

Required parameters

-dataSourceName
The name of the data source. The default value is `jdbc/wimDS`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-databaseType
The type of the database. The default value is `DB2`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-dbURL
The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-dbAdminId
The database administrator ID. (String, required if database type is not Apache Derby.)

-dbAdminPassword
The database administrator password. (String, required if database type is not Apache Derby.)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-JDBCDriverClass
The JDBC driver class name. (String, optional)

-dbSchema
The database schema of the database repository that you want to configure. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user. (String, optional).

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrEntry MappingRepository {-dbAdminId database_administrator_ID -dbAdminPassword database_administrator_password}
```

- Using Jython string:

```
AdminTask.setIdMgrEntry MappingRepository ('[-dbAdminId database_administrator_ID -dbAdminPassword database_administrator_password]')
```

- Using Jython list:

```
AdminTask.setIdMgrEntry MappingRepository (['-dbAdminId', 'database_administrator_ID', '-dbAdmin Password', 'database_administrator_password'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrEntryMapping Repository {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrEntryMapping Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrEntryMapping Repository (['-interactive'])
```

setIdMgrPropertyExtensionRepository

The **setIdMgr Property Extension Repository** command sets or updates the property extension repository configuration.

Important: The application server cannot validate the data source when you run this command in the local mode.

Required parameters

-dataSourceName
The name of the data source. The default value is `jdbc/wimDS`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-databaseType
The type of the database. The default value is `DB2`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-dbURL
The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)

-dbAdminId
The database administrator ID. (String, required if database type is not Apache Derby.)

-dbAdminPassword
The database administrator password. (String, required if database type is not Apache Derby.)

-entityRetrievalLimit
The limit for the retrieval of entities. (Integer, required)

-JDBCDriverClass
The JDBC driver class name. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-dbSchema
The database schema of the database repository that you want to configure. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user. (String, optional).

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrProperty ExtensionRepository {-entity RetrievalLimit limit_value -JDBC DriverClass class_name}
```

- Using Jython string:

```
AdminTask.setIdMgrProperty ExtensionRepository ('[-entity RetrievalLimit limit_value -JDBC DriverClass class_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrProperty ExtensionRepository (['-entity RetrievalLimit', 'limit_value', '-JDBCDriverClass', 'class_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrProperty ExtensionRepository {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrProperty ExtensionRepository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrPropertyExt ensionRepository (['-interactive'])
```

updateIdMgrDBRepository

The **updateId MgrDB Repository** command updates the configuration for the database repository that you specify.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-dataSourceName
The name of the data source. The default value is `jdbc/wimDS`. (String, optional)

-databaseType
The type of the database. The default value is `DB2`. (String, optional)

-dbURL
The URL of the database. (String, optional)

-dbAdminId
The database administrator ID. (String, optional)

-dbAdminPassword
The database administrator password. (String, optional)

-entityRetrievalLimit
Indicates the value of the retrieval limit on database entries. The default value is `200`. (Integer, optional)

-JDBCDriverClass
The JDBC driver class name. (String, optional)

-saltLength
The salt length in bits. The default value is `12`. (Integer, optional)

-encryptionKey
The default value is `rZ15ws0ely9yHk3zCs3sTMv/ho8fY17s`. (String, optional)

-dbSchema
The database schema of the database repository that you want to configure. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user. (String, optional).

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrDB Repository {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrDB Repository ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrDB Repository (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrDB Repository {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrDB Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrDB Repository (['-interactive'])
```

updateIdMgrFileRepository

The **updateId MgrFile Repository** command updates the configuration for the file repository that you specify. To update other properties of the file repository use the **update IdMgr Repository**command.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-messageDigest Algorithm
The message digest algorithm that will be used for hashing the password. The default value is `SHA-1`. Valid values include the following: `SHA-1`, `SHA-384`, or `SHA-512`.(String, optional)

-baseDirectory
The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional)

-fileName
The file name of the repository. The default value is `fileRegistry.xml`. (String, optional)

-saltLength
The salt length of the randomly generated salt for password hashing. The default value is `12`. (Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrFile Repository {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrFile Repository ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrFile Repository (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrFile Repository {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrFile Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrFile Repository (['-interactive'])
```

updateIdMgrLDAPAttrCache

The **updateId MgrLDAP AttrCache** command updates the LDAP attribute cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cachesDiskOffLoad
(String, optional)

-enabled
Indicates if you want to enable attribute caching. The default value is `true`. (Boolean, optional)

-cacheSize
The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is `100`. The default value is `4000`. (Integer, optional)

-cacheTimeOut
The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is `0`. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is `1200`. (Integer, optional)

-attributeSizeLimit
An integer that represents the maximum number of attribute object values that can cache in the attributes cache.

Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is `2000`. (Integer, optional)

-serverTTLAttribute
The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out.

The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of `0` implies that the object will not be cached. For more information about this attribute, go to: http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt.

The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttyl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.

-cacheDistPolicy
The distribution policy for the dynamic cache in a cluster environment.

The valid values are `none` (for NOT_SHARED), `push` (for SHARED_PUSH), and `push_pull` (for SHARED_PUSH_PULL) and the default value is `none`. The value of this parameter is read during the adapter startup process and the cache policy is set accordingly.(String, optional)

For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional)

- Returns: None

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP AttrCache {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP AttrCache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP AttrCache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP AttrCache {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP AttrCache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP AttrCache (['-interactive'])
```

updateIdMgrLDAPContextPool

The **updateId MgrLDAP ContextPool** command updates the LDAP context pool configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-enabled
By default, the context pool is enabled. If you set the value of this parameter to `false`, the context pool is disabled which means that a new context instance will be created for each request. The default value is `true`. (Boolean, optional)

-initPoolSize
The number of context instances that the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is `1` to `50`. The default value is `1`. (Integer, optional)

-maxPoolSize
The maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number. When the pool size reaches this number, new context instances cannot created for new request. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the value defined for the poolWaitTime parameter. The minimum value of the maxPoolSize parameter is `0`. There is no maximum value. A

maximum pool size of 0 means that there is no maximum size and that a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is `0`. (Integer, optional)

-prefPoolSize
The preferred number of context instances that the Context Pool should maintain. Both in-use and idle context instances contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, Context Pool will create and use a new pooled context instance regardless of whether an idle connection is available. When a request is finished with a pooled context instance and the pool size is greater than the preferred size, the Context Pool will close and remove the pooled context instance from the pool. The valid range of the prefPoolSize parameter is `0` to `100`. A preferred pool size of 0 means that there is no preferred size: A request for a pooled context instance will result in a newly created context instance only if no idle ones are available. The default value is `3`. (Integer, optional)

-poolTimeOut
An integer that represents the number of seconds that an idle context instance may remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection will be closed no matter this context instance is stale or active. A new context instance will be created and put back to the pool after it has been released from the request.The minimum value of poolTimeout is 0. There is no maximum value.A poolTimeout of 0 means that the context instances in the pool will remain in the pool until they are staled. In this case, Context Pool will catch the communication exception and recreate a new context instance. The default value is `0`. (Integer, optional)

-poolWaitTime
The time interval (in milliseconds) that the request will wait until the Context Pool checks again if there are idle context instance available in the pool when the number of context instances reaches the maximum pool size. If there is still no idle context instance, the request will continue waiting for the same period of time until next checking. The minimum value of poolWaitout is 0. There is no maximum value. A poolWaitTime of 0 means the Context Pool will not check if there are idle context. Instead, the request will be notified when there is a context instance is released from other requests. The default value is `3000`. (Integer, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP ContextPool {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP ContextPool ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP ContextPool (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP ContextPool {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP ContextPool ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP ContextPool (['-interactive'])
```

updateIdMgrLDAPEntityType

The **updateId MgrLDAP EntityType** command updates an existing LDAP entity type definition to LDAP repository configuration. You can use this command to add more values to multi-valued parameters. If the property already exists, the value of the property will be replaced. If the property does not exist, it will be added.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the entity type. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-searchFilter
The search filter that you want to use to search the entity type. (String, optional)

-objectClasses
One or more object classes for the entity type. (String, optional)

-objectClassesForCreate
The object class that will be when you create an entity type object. You do not have to specify the value of this parameter if it is the same as the value of the objectClasses parameter. (String, optional)

-searchBases
The search base or bases to use while searching the entity type. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAPEntity Type {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPEntity Type ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPEntity Type (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP EntityType {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP EntityType ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP EntityType (['-interactive'])
```

updateIdMgrLDAPGroupDynamicMemberAttr

The **updateIdMgr LDAPGroup Dynamic MemberAttr** command updates a dynamic member attribute configuration to an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute. For example, `memberURL`. (String, required)

-objectClass
The group object class that contains the dynamic member attribute. For example `groupOfURLs`. If you do not define this parameter, the dynamic member attribute will apply to all group object classes. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-id id_name -name name_value -objectClass groupOfURLs}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr ('[-id id_name -name name_value -objectClass groupOfURLs]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-id', 'id_name', '-name', 'name_value', '-objectClass', 'groupOfURLs'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])
```

updateIdMgrLDAPGroupMemberAttr

The **updateIdMgr LDAPGroup MemberAttr** command updates a member attribute configuration of an LDAP group configuration.

Required parameters

-id
The ID of the repository. (String, required)

-name
The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-objectClass
The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUniqueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional)

-scope
The scope of the member attribute. The following are the valid values:

- direct - The member attribute only contains direct members whereby the member is directly contained by the group and not contained in a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct member of group1 but user1 is not a direct member of group1. Both member and uniqueMember are direct member attributes.
- nested - The member attribute contains both direct members and nested members.

-dummyMember
When you create a group without specifying a member, a dummy member will be filled in automatically to avoid receiving an exception that indicates that there is a mandatory attribute missing. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP GroupMemberAttr {-id id_name -name name_value}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP GroupMemberAttr ('[-id id_name -name name_value]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP GroupMemberAttr (['-id', 'id_name', '-name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAPGroup MemberAttr {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPGroup MemberAttr ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPGroup MemberAttr (['-interactive'])
```

updateIdMgrLDAPRepository

The **updateId MgrLDAP Repository** command updates an LDAP repository configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-ldapServerType
The type of LDAP server that is being used. The default value is `IDS51`. (String, optional)

-adapterClassName
The default value is `com.ibm.ws. wim.adapter. ldap.LdapA dapter`. (String, optional)

-certificateMapMode
Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify `certificatefilter`. (String, optional)

-certificateFilter
If `certificateMapMode` has the value `certificatefilter`, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional)

-isExtIdUnique
Specifies if the external ID is unique. The default value is `true`. (Boolean, optional)

-loginProperties
Indicates the property name used for login. (String , optional)

Note: If you define multiple login properties, then the first login property is programmatically mapped to the federated repositories principalName property. For example, if you set `uid;mail` as the login properties, the LDAP attribute uid value is mapped to the federated repositories principalName property. If you define multiple login properties, after login, the first login property is returned as the value of the principalName property. For example, if you pass `joe@yourco.com` as the principalName value and the login properties are configured as uid;mail, the principalName is returned as joe.

-primaryServerQueryTimeInterval
Indicates the polling interval for testing the primary server availability. The value of this parameter is specified in minutes. The default value is `15`. (Integer, optional)

-returnToPrimaryServer
Indicates to return to the primary LDAP server when it is available. The default value is `true`. (Boolean, optional)

-searchCountLimit
The value of search count limit. (Integer, optional)

-searchPageSize
The value of search page size. (Integer, optional)

-searchTimeLimit
The value of search time limit. (Integer, optional)

-sslConfiguration
The SSL configuration. (String, optional)

-supportAsyncMode
Indicates if the async mode is supported or not. The default value is `false`. (Boolean, optional)

-supportChangeLog
This parameter indicates whether the repository supports change tracking. Valid values for this parameter are none or native. The default value is none. (String, optional)

-supportSorting
Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional)

-supportPaging
Indicates if paging is supported or not. The default value is `false`. (Boolean, optional)

-supportTransactions
Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional)

-supportExternalName
Indicates if external names are supported or not. The default value is `false`. (Boolean, optional)

-translateRDN
Indicates to translate RDN or not. The default value is `false`. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP Repository {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP Repository ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP Repository (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP Repository {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP Repository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP Repository (['-interactive'])
```

updateIdMgrLDAPSearchResultCache

The **updateIdMgr LDAPSearch ResultCache** command updates the LDAP search result cache configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cachesDiskOffLoad
Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is `false`. (Boolean, optional)

-enabled
Enables the search results cache. The default value is `true`. (Boolean, optional)

-cacheSize
The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is `100`. The default value is `2000`. (Integer, optional)

-cacheTimeOut
The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is `0`. A value of `0` means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is `600`. (Integer, optional)

-searchResultSizeLimit
The maximum number of entries contained in the naming enumeration object that can be cached in

the search results cache.For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to `1000`. The default value is `1000`. (Integer, optional)

-cacheDistPolicy
The distribution policy for the dynamic cache in a cluster environment.

The valid values are `none` (for NOT_SHARED), `push` (for SHARED_PUSH), and `push_pull` (for SHARED_PUSH_PULL) and the default value is `none`. The value of this parameter is read during the adapter startup process and the cache policy is set accordingly.(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP SearchResultCache {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPSearch ResultCache ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPSearch ResultCache (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP SearchResultCache {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPSearch ResultCache ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPSearch ResultCache (['-interactive'])
```

updateIdMgrLDAPServer

The **updateIdMgr LDAPServer** command updates an LDAP server configuration for the LDAP repository ID that you specify.

Required parameters and return values

-id
The ID of the repository. (String, required)

-host
The host name for the LDAP server that contains the properties that you want to modify. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-port
The port number for the LDAP server. (Integer, optional)

-authentication
Indicates the authentication method to use. The default value is `simple`. Valid values include: `none` or `strong`. (String, optional)

-bindDN
The binding domain name for the LDAP server. (String, optional)

-bindPassword
The binding password. The password is encrypted before it is stored.(String, optional)

-certificateMapMode
Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify `certificatefilter`. (String, optional)

-certificateFilter
If `certificateMapMode` has the value `certificatefilter`, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional)

-connectTimeout
The connection timeout measured in seconds. (Integer, optional)

Restriction: Due to a current JNDI limitation, the maximum connection timeout is 20 seconds. Even if you specify a value above 20 seconds, the connection still times out at 20 seconds.

-connectionPool
The connection pool. The default value is `false`. (Boolean, optional)

-derefAliases
Controls how aliases are dereferenced. The default value is `always`. Valid values include:

- `never` - never deference aliases
- `finding` - deferences aliases only during name resolution
- `searching` - deferences aliases only after name resolution

(String, optional)

-ldapServerType
The type of LDAP server being used. The default value is `IDS51`. (String, optional)

-primary_host
The host name for the primary LDAP server. (String, optional)

-referal
The LDAP referral. The default value is `ignore`. Valid values include: `follow`, `throw`, or `false`. (String, optional)

-sslConfiguration
The SSL configuration. (String, optional)

-sslEnabled
Indicates to enable SSL or not. The default value is `false`. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAPServer {-id id_name -host myhost.ibm.com}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAPServer ('[-id id_name -host myhost.ibm.com]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAPServer (['-id', 'id_name', '-host', 'myhost.ibm.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrLDAP Server {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrLDAP Server ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrLDAP Server (['-interactive'])
```

updateIdMgrRepository

The **updateIdMgr Repository** command updates the common repository configuration.

Required parameters

-id
The ID of the repository. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-adapterClassName
The implementation class name for the repository adapter. (String, optional)

-EntityTypesNot AllowCreate
The name of the entity type that should not be created in this repository. (String, optional)

-EntityTypesNotAllowUpdate
The name of the entity type that should not be updated in this repository. (String, optional)

-EntityTypesNotAllowRead
The name of the entity type that should not be read from this repository. (String, optional)

-EntityTypesNotAllowDelete
The name of the entity type that should not be deleted from this repository. (String, optional)

-isExtIdUnique
Specifies if the external ID is unique or not. (Boolean, optional)

-loginProperties
Indicates the property name used for login. (String, optional)

-readOnly
Indicates if this is a read only repository. The default value is `false`. (Boolean, optional)

-repositoriesForGroups
The repository ID where group data is stored. (String, optional)

-supportAsyncMode
Indicates if the adapter supports async mode or not. The default value is `false`. (Boolean, optional)

-supportChangeLog
This parameter indicates whether the repository supports change tracking. Valid values for this parameter are none or native. The default value is none. (String, optional)

-supportPaging
Indicates if the repository supports paging or not. (Boolean, optional)

-supportSorting
Indicates if the repository supports sorting or not. (Boolean, optional)

-supportTransactions
Indicates if the repository supports transaction or not. (Boolean, optional)

-supportedExternalName
Indicates if the repository supports external names or not. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrRe pository {-id id_name}
```

- Using Jython string:

```
AdminTask.updateIdMgrRe pository ('[-id id_name]')
```

- Using Jython list:

```
AdminTask.updateIdMgr Repository (['-id', 'id_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrRe pository {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrRe pository ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrRe pository (['-interactive'])
```

updateIdMgrRepositoryBaseEntry

The **updateIdMgr Repository BaseEntry** command updates a base entry to the specified repository.

Required parameters

-id
The ID of the repository. (String, required)

-name
The distinguished name of a base entry. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-nameInRepository
The distinguished name in the repository that uniquely identifies the base entry name. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrReposi toryBaseEntry {-id id_name name name_value}
```

- Using Jython string:

```
AdminTask.updateIdMgrRepository BaseEntry ('[-id id_name name name_value]')
```

- Using Jython list:

```
AdminTask.updateIdMgrReposi toryBaseEntry (['-id', 'id_name', 'name', 'name_value'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrRepositoryBaseEntry {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrRepositoryBaseEntry ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrRepositoryBaseEntry (['-interactive'])
```

IdMgrRealmConfig command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure federated repositories realms. The commands and parameters in the IdMgrRealmConfig group can be used to create and manage your realm configuration.

The IdMgrRealmConfig command group for the AdminTask object includes the following commands:

addIdMgrRealmBaseEntry

The **addIdMgrRealmBaseEntry** command adds a base entry to a specific realm configuration and links the realm with the repository.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

-baseEntry
Specifies the name of the base entry. (String, optional)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrRealmBaseEntry {-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository}
```

- Using Jython string:

```
AdminTask.addIdMgrRealmBaseEntry ('[-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository]')
```

- Using Jython list:

```
AdminTask.addIdMgrRealmBaseEntry (['-name', 'defaultWIMFileBasedRealm', '-baseEntry', 'o=sampleFileRepository'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrRealmBaseEntry {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrRealmBaseEntry ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrRealmBaseEntry (['-interactive'])
```

createIdMgrRealm

The **createIdMgrRealm** command creates a realm configuration.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-securityUse
Specifies a string that indicates if this virtual realm will be used in security now, later, or never. The default value is `active`. Additional values includes: `inactive` and `nonSelectable`. (String, optional)

-delimiter
Specifies the delimiter used for this realm. The default value is `/`. (String, optional)

-allowOperationIfReposDown
Specifies whether the system allows a repository operation such as `get` or `search` to complete successfully, even if repositories in the realm are down. The default value is `false`. (Boolean, optional)

gotcha: Even if this parameter is specified, all repositories must be available when you start the server, or the federated repositories will not function properly.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrRealm {-name realm1 -allowOperationIfReposDown true}
```

- Using Jython string:

```
AdminTask.createIdMgrRealm ('[-name realm1 -allowOperationIfReposDown true]')
```

- Using Jython list:

```
AdminTask.createIdMgrRealm (['-name', 'realm1', '-allowOperationIfReposDown', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createIdMgrRealm {-interactive}
```

- Using Jython string:

```
AdminTask.createIdMgrRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createIdMgrRealm (['-interactive'])
```

deleteIdMgrRealm

The **deleteIdMgrRealm** command deletes the realm configuration that you specified.

Target Object

None.

Required parameters

-name
The realm name. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRealm {-name realm1}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRealm ('[-name realm1]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRealm (['-name', 'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRealm {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRealm (['-interactive'])
```

deleteIdMgrRealmBaseEntry

The **deleteIdMgrRealmBaseEntry** command deletes a base entry from a realm configuration that you specified.

The realm must always contain at least one base entry, thus you cannot remove every entry.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

-baseEntry
Specifies the name of a base entry. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

ne.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRealmBaseEntry {-name realm1 -baseEntry entry1}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRealmBaseEntry ('[-name realm1 -baseEntry entry1]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRealmBaseEntry (['-name', 'realm1', '-baseEntry', 'entry1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrRealmBaseEntry {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrRealmBaseEntry ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrRealmBaseEntry (['-interactive'])
```

getIdMgrDefaultRealm

The **getIdMgrDefaultRealm** command returns the default realm name.

Target Object

None.

Required parameters

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrDefaultRealm
```

- Using Jython string:

```
AdminTask.getIdMgrDefaultRealm()
```

- Using Jython list:

```
AdminTask.getIdMgrDefaultRealm()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrDefaultRealm {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrDefaultRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrDefaultRealm (['-interactive'])
```

getIdMgrRepositoriesForRealm

The **getIdMgrRepositoriesForRealm** command returns repository specific details for the repositories configured for a specified realm.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRepositoriesForRealm {-name realm1}
```

- Using Jython string:

```
AdminTask.getIdMgrRepositoriesForRealm ('[-name realm1]')
```

- Using Jython list:

```
AdminTask.getIdMgrRepositoriesForRealm (['-name', 'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRepositoriesForRealm {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrRepositoriesForRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrRepositoriesForRealm (['-interactive'])
```

getIdMgrRealm

The **getIdMgrRealm** command returns the configuration parameters for the realm that you specified.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRealm {-name realm1}
```

- Using Jython string:

```
AdminTask.getIdMgrRealm ('[-name realm1]')
```

- Using Jython list:

```
AdminTask.getIdMgrRealm (['-name', 'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getIdMgrRealm {-interactive}
```

- Using Jython string:

```
AdminTask.getIdMgrRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getIdMgrRealm (['-interactive'])
```

listIdMgrRealms

The **listIdMgrRealms** command returns all of the names of the configured realms.

Target Object

None.

Required parameters

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealms
```

- Using Jython string:

```
AdminTask.listIdMgrRealms()
```

- Using Jython list:

```
AdminTask.listIdMgrRealms()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealms {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrRealms ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrRealms (['-interactive'])
```

listIdMgrRealmBaseEntries

The **listIdMgrRealmBaseEntries** command returns all of the names of the configured realms.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealmBaseEntries {-name realm1}
```

- Using Jython string:

```
AdminTask.listIdMgrRealmBaseEntries ('[-name realm1]')
```

- Using Jython list:

```
AdminTask.listIdMgrRealmBaseEntries (['-name', 'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealmBaseEntries {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrRealmBaseEntries ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrRealmBaseEntries (['-interactive'])
```

listIdMgrRealmURAttrMappings

Use the listIdMgrRealmURAttrMappings command to list the mappings between the user or group attributes for a user registry and the federated repository properties of a specified realm.

Target object

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-name
Use this parameter to specify a valid realm name for which you want to list the mapping.

If you do not specify the **-name** parameter, the listIdMgrRealmURAttrMappings command returns the mapping of the default realm in the federated repository configuration.

(String, optional)

Return values

The listIdMgrRealmURAttrMappings command returns a HashMap that contains the following structure:

- The key is the user registry attribute name (**URAttrName** parameter).
- The value is another HashMap that contains the **propertyForInput** and **propertyForOutput** as keys and the corresponding mapping as the values.

The following example shows a sample output. The example is broken into multiple lines for illustration purposes only.

```
{userDisplayName={propertyForInput=principalName, propertyForOutput=principalName},
userSecurityName={propertyForInput=principalName, propertyForOutput=principalName},
uniqueUserId={propertyForInput=uniqueName, propertyForOutput=uniqueName},
uniqueGroupId={propertyForInput=uniqueName, propertyForOutput=uniqueName},
groupSecurityName={propertyForInput=cn, propertyForOutput=cn},
groupDisplayName={propertyForInput=cn, propertyForOutput=cn}}
```

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealmURAttrMappings
```

- Using Jython string:

```
AdminTask.listIdMgrRealmURAttrMappings()
```

- Using Jython list:

```
AdminTask.listIdMgrRealmURAttrMappings()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrRealmURAttrMappings {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrRealmURAttrMappings ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrRealmURAttrMappings (['-interactive'])
```

renameIdMgrRealm

The **renameIdMgrRealm** command renames the name of the realm that you specified.

Note: Renaming the federated repositories realm name does not update the realm name stored in the `security.xml` file.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

-newName
Specifies the new name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask renameIdMgrRealm {-name realm1 -newName realm2}
```

- Using Jython string:

```
AdminTask.renameIdMgrRealm ('[-name realm1 -newName realm2]')
```

- Using Jython list:

```
AdminTask.renameIdMgrRealm (['-name', 'realm1', '-newName', 'realm2'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask renameIdMgrRealm {-interactive}
```

- Using Jython string:

```
AdminTask.renameIdMgrRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.renameIdMgrRealm (['-interactive'])
```

setIdMgrDefaultRealm

The **setIdMgrDefaultRealm** command sets the default realm name.

Required parameters

-name
Specifies the name of the realm that is used as a default realm when the caller does not specify any in context. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrDefaultRealm {-name realm1}
```

- Using Jython string:

```
AdminTask.setIdMgrDefaultRealm ('[-name realm1]')
```

- Using Jython list:

```
AdminTask.setIdMgrDefaultRealm (['-name', 'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrDefaultRealm {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrDefaultRealm ('[-interactive]')
```

setIdMgrRealmURAttrMapping

Use the setIdMgrRealmURAttrMapping command to set or modify the mapping of the user or group attribute for a user registry to a federated repository property of a specified realm.

The setIdMgrRealmURAttrMapping command is available in both connected and local modes. If you run the setIdMgrRealmURAttrMapping command in connected mode, the realm attribute mapping changes take effect after you restart the server.

Target object

None.

Required parameters

-URAttrName
Use this parameter to specify the name of the user or group attribute in a user registry that you want to map. The following case-sensitive values are valid for the **URAttrName** parameter:

- uniqueUserId
- userSecurityName
- userDisplayName
- uniqueGroupId
- groupSecurityName
- groupDisplayName

Note: If you run the setIdMgrRealmURAttrMapping command multiple times for the same user registry attribute name, it overwrites the previous value.

(String, required)

-propertyForInput
Use this parameter to specify the name of the federated repository property that maps to the specified user registry attribute (**URAttrName** parameter) when it is an input parameter for the user registry interface. (String, required)

-propertyForOutput
Use this parameter to specify the name of the federated repository property that maps to the specified user registry attribute (**URAttrName** parameter) when it is an output parameter (return value) for the user registry interface. (String, required)

Attention: In most cases, the **propertyForInput** and **propertyForInput** would be the same.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-name
Use this parameter to specify a valid realm name for which you want to set or modify the mapping. If you do not specify the name parameter, the setIdMgrRealmURAttrMapping command uses the default realm in the federated repository configuration. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrRealmURAttrMapping {-URAttrName unique_user_ID -propertyForInput unique_name -propertyForOutput unique_name}
```

- Using Jython string:

```
AdminTask.setIdMgrRealmURAttrMapping ('[-URAttrName unique_user_ID -propertyForInput unique_name -propertyForOutput unique_name]')
```

- Using Jython list:

```
AdminTask.setIdMgrRealmURAttrMapping (['-URAttrName', 'unique_user_ID', '-propertyForInput', 'unique_name', '-propertyForOutput', 'unique_name'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask setIdMgrRealmURAttrMapping {-interactive}
```

- Using Jython string:

```
AdminTask.setIdMgrRealmURAttrMapping ('[-interactive]')
```

- Using Jython list:

```
AdminTask.setIdMgrRealmURAttrMapping (['-interactive'])
```

updateIdMgrRealm

The **updateIdMgrRealm** command updates the configuration for a realm that you specify.

Target Object

None.

Required parameters

-name
Specifies the name of the realm. (String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-securityUse
Specifies a string that indicates if this realm will be used in security now, later, or never. The default value is `active`. Additional values includes: `inactive` and `nonSelectable`. (String, optional)

-delimiter
specifies the delimiter used for this realm. The default value is /. (String, optional)

-allowOperationIfReposDown
Specifies whether the system allows a repository operation such as `get` or `search` to complete successfully, even if repositories in the realm are down. (Boolean, optional)

gotcha: Even if this parameter is specified, all repositories must be available when you start the server, or the virtual member manager might not function properly.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrRealm {-name
realm1}
```

- Using Jython string:

```
AdminTask.updateIdMgrRealm ('[-name
realm1]')
```

- Using Jython list:

```
AdminTask.updateIdMgrRealm (['-name',
'realm1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateIdMgrRealm {-interactive}
```

- Using Jython string:

```
AdminTask.updateIdMgrRealm ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateIdMgrRealm (['-interactive'])
```

IdMgrDataModel command group for the AdminTask object

You can use the Jython or Jacl scripting language to manage the federated repository schema using the wsadmin tool. Use the commands and parameters in the IdMgrDataModel group to manage the property extension repository. The commands are available in connected or local mode using the –conntype NONE option.

The IdMgrDataModel command group for the AdminTask object includes the following commands:

- "addIdMgrPropertyToEntityTypes"
- "listIdMgrPropertyExtensions" on page 466

addIdMgrPropertyToEntityTypes

Use the addIdMgrPropertyToEntityTypes command to add a new property to one or more existing entity types with specified parameters.

Target Object

None.

Required parameters

-name
Use this parameter to specify the name of the new property that is added to one or more existing entity types.

(String, required)

-dataType
Use this parameter to specify the data type of the property. The default supported data types are:

- String
- Integer
- Long
- Double
- DateTime
- Base64Binary

- Identifier
- Object

(String, required)

-entityTypeNames
This parameter specifies the name of one or more existing entity types to which the new property is added. Use a semicolon (;) as the delimiter to specify multiple entity types.

You can use a valid namespace prefix (nsPrefix) to refer to the entity type names in a specific namespace. The default namespace prefix is wim. Use a colon (:) to specify the namespace prefix. For example, you can use: *your_ext*:*Person_Account*

- *your_ext* is the namespace prefix, which is mapped to the http://www.yourco.com/yourext namespace URI.
- *Person_Account* is the entity type name.

(String, required)

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-nsURI
Use this parameter to specify the namespace to which the new property is added.

You must map each namespace URI (nsURI) value to a unique namespace prefix (nsPrefix) value the first time that you use the namespace URI. The new property is added to the default namespace if you do not specify a nsURI value. The default namespace is http://www.ibm.com/websphere/wim, which is mapped, by default, to the wim nsPrefix value.

(String, optional)

-nsPrefix
Use this parameter to specify the prefix for the namespace.

You must specify a namespace prefix (nsPrefix) value, if you specify a new namespace URI (nsURI) value, so that the new nsURI value is mapped to this nsPrefix value. The default nsPrefix value is wim, which is mapped, by default, to the http://www.ibm.com/websphere/wim nsURI value.

(String, optional)

-isMultiValued
Use this parameter to specify whether the new property accepts more than one value. The default value is false.

(Boolean, optional)

-repositoryIds
Use this parameter to specify the repository or repositories to which the new property is added.

Use a semicolon (;) as the delimiter to specify multiple repository identities. All of the configured repositories are included, if this parameter is not specified.

To add the new property into the property extension repository, specify `LA` as the value of this parameter.

To add the new property to the entity types for all the specified repositories, specify a list of repository identities and do not include LA in the list.

To extend the property to the property extension repository for particular Lightweight Directory Access Protocol (LDAP) repositories, specify a list that includes LA. The new extended property is then marked as not supported for the other LDAP repositories that are included in this list.

(String, optional)

-requiredEntityTypeNames

Use this parameter to specify one or more entity types for which the new property is required.

Use a semicolon (;) as the delimiter to specify multiple entity types.

You can use a valid namespace prefix (nsPrefix) to refer to the entity type names in a specific namespace. The default namespace prefix is wim. Use a colon (:) to specify the namespace prefix. For example, you can use: *your_ext*:*Person_Account*

- *your_ext* is the namespace prefix, which is mapped to the http://www.yourco.com/yourext namespace URI.
- *Person_Account* is the entity type name.

For example, if you are adding the contactNumber property to the PersonAccount and Group entity types and you require that all PersonAccount entity types have a contactNumber value, then specify:

```
{-name contactNumber -entityTypeNames PersonAccount;Group  -requiredEntityTypeNames PersonAccount}
```

(String, optional)

-applicationId

Use this parameter to indicate the ID of the application for which the property is extended.

You can use the “listIdMgrPropertyExtensions” command to retrieve the extended properties for a specific application ID, if you specify a value for the **applicationId** parameter.

(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrPropertyToEntityTypes {-name property_name_1 -dataType string -entityTypeNames entity_type_1;entity_type_2}
```

- Using Jython string:

```
AdminTask.addIdMgrPropertyToEntityTypes ('[-name property_name_1 -dataType string -entityTypeNames entity_type_1;entity_type_2]')
```

- Using Jython list:

```
AdminTask.addIdMgrPropertyToEntityTypes (['-name', 'property_name_1', '-dataType', 'string' '-entityTypeNames', 'entity_type_1;entity_type_2'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addIdMgrPropertyToEntityTypes {-interactive}
```

- Using Jython string:

```
AdminTask.addIdMgrPropertyToEntityTypes ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addIdMgrPropertyToEntityTypes (['-interactive'])
```

listIdMgrPropertyExtensions

Use the listIdMgrPropertyExtensions command to list the properties extended for entity types.

Target Object

None.

Required parameters

None.

Optional parameters

-securityDomainName
Use this parameter to specify the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-applicationId
Use this parameter to indicate the ID of the application for which the extended properties are listed.

If you do not specify this parameter, all the extended properties are listed.

(String, optional)

Return value

This parameter returns a hash map that contains the property name as the key. The value of each key is another hash map, which contains the same keys as the input parameters of the "addIdMgrPropertyToEntityTypes" on page 464 command. For multivalued parameters, the value of the key is a List object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrPropertyExtensions
```

- Using Jython string:

```
AdminTask.listIdMgrPropertyExtensions()
```

- Using Jython list:

```
AdminTask.listIdMgrPropertyExtensions()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listIdMgrPropertyExtensions {-interactive}
```

- Using Jython string:

```
AdminTask.listIdMgrPropertyExtensions ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listIdMgrPropertyExtensions (['-interactive'])
```

Note: This command lists the extended properties that are present in the `wimxmlextension.xml` file only.

IdMgrDBSetup command group for the AdminTask object

You can use the Jython or Jacl scripting language to manage the federated repository schema using the wsadmin tool. Use the deleteIdMgrPropertyExtensionEntityDatacommand and its parameters in the IdMgrDBSetup group to manage the property extension repository. The command is available in both connected and local mode using the –conntype NONE option.

deleteIdMgrPropertyExtensionEntityData

The deleteIdMgrPropertyExtensionEntityData command deletes property data from the property extension repository. The command also releases the space that is held by the property data in the property extension repository.

Target Object

None.

Required parameters

-name
Use this parameter to specify the name of the property for which the data is to be deleted.

You can use a valid namespace prefix to refer to the property in a specific namespace. The default namespace prefix is `wim`. Use a colon (:) as the delimiter to specify the namespace prefix. For example, you can use *your_ext*:*contact_number*

- *your_ext* is the namespace prefix.
- *contact_number* is the property name.

(String, required)

Important: You must specify the same namespace prefix that you used to extend the property. You can use the listIdMgrPropertyExtensions command, which is part of the IdMgrDataModel command group, to retrieve this value.

Optional parameters

-entityTypeNames
Use this parameter to specify the name of one or more existing entity types for which the property data is to be deleted. Use the semicolon (;) as the delimiter to specify multiple entity types.

You can use a valid namespace prefix to refer to the entity type names in a specific namespace. The default namespace prefix used is `wim`. Use colon (:) to specify the namespace prefix. For example, you can use *your_ext*:*person_account*

- *your_ext* is the namespace prefix.
- *person_account* is the entity type name.

(String, optional)

Important: You must specify the same namespace prefix that you used to extend the property. You can use the listIdMgrPropertyExtensions command, which is part of the IdMgrDataModel command group, to retrieve this value.

-dbAdminId
Use this parameter to specify the login identity of the database administrator when you run this command in the local mode.

(String, optional)

-dbAdminPassword
Use this parameter to specify the password of the database administrator when you run this command in the local mode.

(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrPropertyExtensionEntityData {-name property_name_1}
```

- Using Jython string:

```
AdminTask.deleteIdMgrPropertyExtensionEntityData ('[-name property_name_1]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrPropertyExtensionEntityData (['-name', 'property_name_1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteIdMgrPropertyExtensionEntityData {-interactive}
```

- Using Jython string:

```
AdminTask.deleteIdMgrPropertyExtensionEntityData ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteIdMgrPropertyExtensionEntityData (['-interactive'])
```

JaspiManagement command group for the AdminTask object

Use the commands and parameters in the JaspiManagement command group to manage the configuration of authentication providers.

Note: WebSphere Application Server supports integration of message authentication providers that are compliant with the JASPI for Containers Version 1.0 specification. Use the Jython or JACL scripting languages to configure Java Authentication Service Provider Interface (JASPI) providers with the **wsadmin** tool.

JASPI is a new specification introduced in Java Platform, Enterprise Edition 6 (Java EE 6). It enables third-party security providers to perform authentication of messages for specific messaging runtime environments. JASPI extends the Java Authentication and Authorization Service (JAAS) architecture with standardized programming interfaces to make network messages available for processing by authentication providers.

If you want to use JASPI message authentication services, you must supply an implementation of the required interfaces as defined in the JASPI specification. Read about Developing a custom authentication provider for more information on these interfaces.

Note: WebSphere Application Server Version 8.0 supports only the Servlet Profile as defined in the JASPI specification.

When JASPI authentication providers are configured, and WebSphere Application Server receives a request message, the security runtime environment determines if the target application is configured to use JASPI authentication. If so, the runtime environment invokes the selected authentication provider to validate the received message. Otherwise, authentication of the message request is done according to the authentication mechanism provided by WebSphere Application Server for the appropriate messaging layer.

The JaspiManagement command group includes the following commands:

configureJaspi

The configureJaspi command is used to specify whether the Java Authentication SPI processing is to be enabled or disabled, and to set the default JASPI provider for a given security domain or the global security configuration.

Target object

None.

Required parameters

None.

Optional parameters

-enabled
Specify `true` to enable the JASPI configuration and `false` to disable the configuration.

-defaultProviderName
Specifies the name of an already configured JASPI provider that is to be used as the default JASPI provider for the security domain or global security configuration.

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

The command returns the object name of the JASPI configuration ID that the system creates, and is an instance of javax.management.ObjectName. The value displayed in the console is the JASPI configuration ID.

Batch mode example usage

Using Jython:

```
AdminTask.configureJaspi('[-enabled true -defaultProviderName testProvider]')
```

Using Jacl:

```
$AdminTask configureJaspi {-enabled true -defaultProviderName testProvider}
```

Interactive mode example usage

Using Jython:

```
AdminTask.configureJaspi('-interactive')
```

Using Jacl:

```
$AdminTask configureJaspi -interactive
```

defineJaspiProvider

The defineJaspiProvider command configures a new authentication provider for the given security domain or the global security configuration.

Target object

None.

Required parameters

-providerName
Specifies a name that uniquely identifies the authentication provider.

-className
Specifies the package-qualified name of the class that implements the authentication provider interface (javax.security.auth.message.config.AuthConfigProvider).

Optional parameters

-description
Specifies a textual description of the authentication provider.

-properties
Specifies additional custom configuration properties needed to initialize the authentication provider. This parameter is a list of key/value pairs.

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

The command returns a map of the authentication provider configuration attributes and their values, and is an instance of `java.util.Map<java.lang.String, java.lang.Object>` that contains the authentication provider configuration attributes. The value displayed in the console is the return value of the Map's toString() method.

Batch mode example usage

Note: For clarity, the command is shown below in multiple lines; it must be entered on a single line.

Using Jython:

```
AdminTask.defineJaspiProvider('-providerName jaspi_provider
-className com.ibm.sample.JaspiProvider -description "Sample authentication provider"
-properties [ [debug true] [user admin] ] ')
```

Using Jacl:

```
$AdminTask defineJaspiProvider {  -providerName jaspi_provider
-className com.ibm.JASPIProvider  -description "Sample authentication provider"
-properties {  {debug true} {user admin}  } }
```

Interactive mode example usage

Using Jython:

```
AdminTask.defineJaspiProvider('-interactive')
```

Using Jacl:

```
$AdminTask defineJaspiProvider -interactive
```

displayJaspiProvider

The displayJaspiProvider command displays the configuration data for one or more given authentication providers for the given security domain or the global security configuration.

Target object

None.

Required parameters

-providerName
Specifies the unique name(s) of the authentication provider(s) to be displayed.

Optional parameters

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

The command displays the specified provider(s) along with their configuration attributes and values. The data returned is displayed as an instance of `java.util.Collection<java.util.Map<java.lang.String, java.lang.Object>>`. Each Map instance contains the configuration attributes of an authentication provider. The value displayed in the console is the return value of the Collection's toString() method.

Batch mode example usage

Using Jython:

```
AdminTask.displayJaspiProvider('-providerName jaspi_provider')
```

Using Jacl:

```
$AdminTask displayJaspiProvider {'-providerName jaspi_provider'}
```

Interactive mode example usage

Using Jython:

```
AdminTask.displayJaspiProvider('-interactive')
```

Using Jacl:

```
$AdminTask displayJaspiProvider -interactive
```

displayJaspiProviderNames

The displayJaspiProviderNames command displays the names of authentication providers in the security configuration. When the securityDomainName parameter is provided, only those providers in the given security domain are displayed. When the securityDomainName parameter is not provided, only the names of the providers in the global security configuration are displayed.

When the securityDomainName parameter is provided with the getEffectiveProviderNames parameter set to `true`, the list of authentication provider names from the given security domain and from the global security configuration are displayed as long as JASPI support is enabled.

Note: The combined list does not include duplicate provider names. For example, if the given security domain configuration has provider names of p1, p2 and p3, and the global security configuration has provider names of p3, p4 and p5, the combined list of provider names contains p1, p2, p3, p4 and p5.

When the securityDomainName parameter is provided with the getEffectiveProviderNames parameter set to `false`, only the list of authentication providers for the given security domain are displayed. The getEffectiveProviderNames parameter only has an effect when used with the securityDomainName parameter; it is ignored if the securityDomainName provider is not provided.

Target object

None.

Required parameters

None.

Optional parameters

-getEffectiveProviderNames
Specify `true` to get the list of effective authentication provider names or `false` to get just the list of authentication providers for the security domain configuration. The default value for getEffectiveProviderName is `false`.

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

The command returns a list of JASPI provider names. The data returned is displayed as an instance of `java.util.Collection<java.lang.String>`.

Batch mode example usage

Using Jython:

```
AdminTask.displayJaspiProviderNames()
```

Using Jacl:

```
$AdminTask displayJaspiProviderNames
```

Interactive mode example usage

Using Jython:

```
AdminTask.displayJaspiProviderNames('-interactive')
```

Using Jacl:

```
$AdminTask displayJaspiProviderNames -interactive
```

getJaspiInfo

The getJaspiInfo command displays information about the JASPI configuration for the given security domain or the global security configuration.

Target object

None.

Required parameters

None.

Optional parameters

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

The command returns an indication of whether Java Authentication SPI processing is enabled. If the command is issued for a specific security domain and a value is customized for the domain, the command returns a value to indicate whether JASPI processing is enabled. If the command is issued for a specific domain and a value is not customized for the domain, it returns an empty list to indicate that the configuration is inherited from the global security configuration. The data returned is displayed as an

instance of java.util.Map<java.lang.String, java.lang.Object> that contains the JASPI configuration attributes. The value displayed in the console is the return value of the Map's toString() method.

In addition, this command returns the value configured for the default provider. For example:

```
wsadmin>$AdminTask getJaspiInfo
{defaultProviderName=null, enabled=false}
```

Batch mode example usage

Using Jython:

```
AdminTask.getJaspiInfo()
```

Using Jacl:

```
$AdminTask getJaspiInfo
```

Interactive mode example usage

Using Jython:

```
AdminTask.getJaspiInfo('-interactive')
```

Using Jacl:

```
$AdminTask getJaspiInfo -interactive
```

modifyJaspiProvider

The modifyJaspiProvider command modifies configuration data for a given authentication provider for the given security domain or the global security configuration.

Target object

None.

Required parameters

-providerName
Specifies a name that uniquely identifies the authentication provider.

Optional parameters

-className
Specifies the package-qualified name of the class that implements the authentication provider interface (javax.security.auth.message.config.AuthConfigProvider).

-description
Specifies a textual description of the authentication provider.

-properties
Specifies additional custom configuration properties needed to initialize the authentication provider. This parameter is a list of key/value pairs.

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

An instance of java.util.Map<java.lang.String, java.lang.Object> that contains any modified configuration attributes in the given authentication provider or any given associated authentication modules. The value displayed in the console is the return value of the Map's toString() method.

Batch mode example usage

Using Jython:

```
AdminTask.modifyJaspiProvider('-providerName jaspi_provider
-properties [ [debug false] ]  ')
```

Using Jacl:

```
$AdminTask modifyJaspiProvider { -providerName jaspi_provider
-properties { {debug false} } }
```

Interactive mode example usage

Using Jython:

```
AdminTask.modifyJaspiProvider('-interactive')
```

Using Jacl:

```
$AdminTask modifyJaspiProvider -interactive
```

removeJaspiProvider

The removeJaspiProvider command removes one or more authentication providers from the security configuration for the given security domain or the global security configuration.

Target object

None.

Required parameters

-providerName
Specifies the unique name(s) of the authentication provider(s) to be removed.

Optional parameters

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Return value

None.

Batch mode example usage

To remove a single provider using Jython:

```
AdminTask.removeJaspiProvider('-providerName jaspi_provider')
```

To remove multiple providers using Jython:

```
AdminTask.removeJaspiProvider( '[-providerName [ Provider1;Provider2 ] ]' )
```

To remove a single user using Jacl:

```
$AdminTask removeJaspiProvider {'-providerName jaspi_provider'}
```

To remove multiple providers using Jacl:

```
$AdminTask removeJaspiProvider {-providerName {Provider1 Provider2 } }
```

Interactive mode example usage

Using Jython:

```
AdminTask.removeJaspiProvider('-interactive')
```

Using Jacl:

```
$AdminTask removeJaspiProvider -interactive
```

unconfigureJaspi

The unconfigureJaspi command is used to remove the JASPI configuration and all of its associated providers from a security domain.

Target object

None.

Required parameters

-securityDomainName
Specifies the name of the security domain. The command uses the global security configuration if you do not specify a value for the -securityDomainName parameter.

Optional parameters

None.

Return value

None.

Batch mode example usage

Using Jython:

```
AdminTask.unconfigureJaspi('-securityDomainName domain1')
```

Using Jacl:

```
$AdminTask unconfigureJaspi {-securityDomainName domain1}
```

Interactive mode example usage

Using Jython:

```
AdminTask.unconfigureJaspi('-interactive')
```

Using Jacl:

```
$AdminTask unconfigureJaspi -interactive
```

LTPACommandGroup command group for the AdminTask object

You can use the Jython or Jacl scripting languages to import and export LTPA keys.

Note: Use the importLTPAKeys command to import an LTPA key from a file and add it to the security runtime and configuration. The exportLTAPKeys command exports an LTPA key that is currently being used by the runtime to a file.

The LTPACommandGroup command group for the AdminTask object includes the following commands:

- “importLTPAKeys” on page 477
- “exportLTPAKeys” on page 477

importLTPAKeys

The importLTPAKeys command imports an LTPA key from a file and adds it to the security runtime and configuration. The command takes the key from the file specified and adds it to the ltpa.jceks keystore where LTPA keys are stored. The key then becomes the current LTPA key of the runtime.

Target object

None.

Required parameters

-ltpaKeyFile
Specifies the path to the file that contains the LTPA key that is to be imported into the configuration and runtime. The parameter takes the file path URL format or just the file path. (String, required)

-password
The password of the LTPA key. (String, required)

Optional parameters

None.

Batch mode example usage

- Using Jacl:

```
wsadmin> $AdminTask importLTPAKeys {-ltpaKeyFile c:\temp\ltpa.key -password keypassword }

wsadmin> $AdminTask importLTPAKeys {-ltpaKeyFile file:\temp\ltpa.key -password keypassword }
```

- Using Jython:

```
wsadmin> AdminTask.importLTPAKeys('[-ltpaKeyFile c:\temp\ltpa.key -password keypassword ]')

wsadmin> AdminTask.importLTPAKeys('[-ltpaKeyFile file:\temp\ltpa.key -password keypassword ]')
```

exportLTPAKeys

The exportLTAPKeys command exports an LTPA key that is currently being used by the runtime to a file. If the file where the key is to be stored does not exist, it is created. If it does not exist, then the file is overwritten.

Target object

None.

Required parameters

-ltpaKeyFile
Specifies the path to the file that contains the LTPA key that is to be imported into the configuration and runtime. The parameter takes the file path URL format or just the file path. (String, required)

-password
The password of the LTPA key. (String, required)

Optional parameters

None.

Batch mode example usage

- Using Jacl:

```
wsadmin> $AdminTask exportLTPAKeys {-ltpaKeyFile c:\temp\writeltpa.key -password keypassword }

wsadmin> $AdminTask exportLTPAKeys {-ltpaKeyFile file:\temp\writeltpa.key -password keypassword }
```

- Using Jython:

```
wsadmin> AdminTask.exportLTPAKeys('[-ltpaKeyFile c:\temp\writeltpa.key -password keypassword ]')

wsadmin> AdminTask.exportLTPAKeys('[-ltpaKeyFile file:\temp\writeltpa.key -password keypassword ]')
```

WIMManagementCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the WIMManagementCommands group can be used to create and manage groups, members, and users in the virtual member manager.

Note: If the Use global security settings option is selected for the user realm or the Global federated repositories option is selected as the realm type for the specified domain, the user and group management commands are executed on the federated repository of the admin domain. For example, if you run the createUser command for the specified domain, the user is created in the admin domain. However, configuration changes that are performed on the domain are applied to the security domain-specific configuration.

The WIMManagementCommands command group for the AdminTask object includes the following commands:

addMemberToGroup

The **addMemberToGroup** command adds a member to a group in the virtual member manager. If successful, the addMemberToGroup command returns the unique name of the added member.

Parameters and return values

-memberUniqueName
Specifies the unique name value for the user or group that you want to add to the specified group. This parameter maps to the uniqueName property in virtual member manager.

-groupUniqueName
Specifies the unique name value for the group to which you want to add the user or group that you specifed in the memberUniqueName parameter. This parameter maps to the uniqueName property in virtual member manager.

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addMemberToGroup {-memberUniqueName uid=meyersd cn=users dc=yourco dc=com -groupUniqueName
 cn=admins cn=groups dc=yourco dc=com}
```

- Using Jython string:

```
AdminTask.addMemberToGroup ('[-memberUniqueName uid=meyersd,cn=users,dc=yourco,dc=com groupUniqueName
cn=admins,cn=groups,dc=yourco,dc=com ]')
```

- Using Jython list:

```
AdminTask.addMemberToGroup (['-memberUniqueName', 'uid=meyersd', 'cn=users', 'dc=yourco', 'dc=com',
 'roupUniqueName', 'cn=admins', 'cn=groups', 'dc=yourco', 'dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addMemberToGroup {-interactive}
```

- Using Jython string:

```
AdminTask.addMemberToGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addMemberToGroup (['-interactive'])
```

changeMyPassword

The **changeMyPassword** command allows you to change your password when you are logged into WebSphere Application Server. It requires you to specify your old password and the new password, and then confirm your new password. If your old password is validated successfully, and the new password that you specify exactly matches your confirmation of the new password, then the password is changed.

Important: You can use the changeMyPassword command only for repositories that have a write adapter for federated repositories. It will not work for read-only adapters or the federated repositories user registry bridge that is configured with the local operating system user registry or a custom user registry.

Parameters and return values

-oldPassword
The old password of the user. The value of the oldPassword parameter is validated against the password of the user in the repository. (String, required)

-newPassword
The new password that must be set for the user. (String, required)

-confirmNewPassword
The new password that must be set for the user. The value of the newPassword and confirmNewPassword parameters must match. (String, required)

Note: After you change your password, your old password might continue to remain in effect, allowing you to login using your old password. This happens if both the authentication cache and basic

authentication cache keys are enabled, causing the old password to remain valid according to the value specified for cache timeout or cache size.

You can clear the WebSphere Application Server security cache so that you do not have to wait for the cacheTimeout to expire. To clean entries from the AuthCache, you must use the SecurityAdmin MBeanclearAuthCache methods, clearAuthCache or purgeUserFromCache.

Call one of the following MBean methods on each WebSphere Application Server process that requires the subject of the user to be cleared from the cache. The AuthCache is a cache for each process, so every process (not just the dmgr) that has the user authenticated must have this method called:

•

```
/**
 * clearAuthCache
 */
public void clearAuthCache()
```

•

```
/**
 * purgeUserFromCache
 */
public void purgeUserFromAuthCache(String realm, String userid)
```

The following example shows how you can use wsadmin to call the clearAuthCache method on the dmgr process:

```
set sa [$AdminControl queryNames type=SecurityAdmin,process=dmgr,*]
$AdminControl invoke $sa clearAuthCache
```

For more information, read Authentication cache settings.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask changeMyPassword {-oldPassword pwd1 –newPassword pwd2
–confirmNewPassword pwd2}
```

- Using Jython string:

```
AdminTask.changeMyPassword ('[oldPassword pwd1 –newPassword pwd2
–confirmNewPassword pwd2]')
```

- Using Jython list:

```
AdminTask.changeMyPassword (['oldPassword', 'pwd1', '–newPassword', 'pwd2',
'–confirmNewPassword', 'pwd2'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask changeMyPassword {-interactive}
```

- Using Jython string:

```
AdminTask.changeMyPassword ('[-interactive]'
```

- Using Jython list:

```
AdminTask.changeMyPassword (['-interactive']))
```

createGroup

The **createGroup** command creates a new group in the virtual member manager. After the command completes, the new group will appear in the repository. For LDAP, a group must contain a member. The memberUniqueName parameter is optional in this case. If you set the memberUniqueName parameter to the unique name of a group or a user, the group or user will be added as a member of the group.

Parameters and return values

-cn
Specifies the common name for the group that you want to create. This parameter maps to the cn property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-description
Specifies additional information about the group that you want to create. This parameter maps to the description property in a virtual member manager object. (String, optional)

-parent
Specifies the repository in which you want to create the group. This parameter maps to the parent property in the virtual member manager. (String, optional)

-memberUniqueName
Specifies the unique name value for the user or group that you want to add to the new group. This parameter maps to the uniqueName property in the virtual member manager. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createGroup {-cn groupA -description a group of admins}
```

- Using Jython string:

```
AdminTask.createGroup ('[-cn groupA -description a group of admins]')
```

- Using Jython list:

```
AdminTask.createGroup (['-cn', 'groupA', '-description', 'a group of admins'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createGroup {-interactive}
```

- Using Jython string:

```
AdminTask.createGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createGroup (['-interactive'])
```

createUser

The **createUser** command creates a new user in the default repository or a repository that the parent command parameter specifies. This command creates a person entity and a login account entity in the virtual member manager.

Parameters and return values

-uid
Specifies the unique ID for the user that you want to create. Virtual member manager then creates a uniqueId value and a uniqueName value for the user. This parameter maps to the uid property in the virutal member manager. (String, required)

-password
Specifies the password for the user. This parameter maps to the password property in the virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-confirmPassword
Specifies the password again to validate how it was entered for the password parameter. This parameter maps to the password property in virtual member manager. (String, optional)

-cn
Specifes the first name or given name of the user. This parameter maps to the cn property in virutal member manager. (String, required)

-sn
Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, required)

-mail
Specifies the email address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. (String, optional)

-parent
Specifies the repository in which you want to create the user. This parameter maps to the parent property in the virtual member manager. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createUser {-uid 123 -password tempPass -confirmPassword tempPass
-cn Jane -sn Doe -mail janedoe@ acme.com}
```

- Using Jython string:

```
AdminTask.createUser ('[-uid 123 -password tempPass -confirmPassword tempPass
 -cn Jane -sn Doe -mail janedoe@ acme.com]')
```

- Using Jython list:

```
AdminTask.createUser (['-uid', '123', '-password', 'tempPass', '-confirmPassword',
'tempPass', '-cn', 'Jane', '-sn', 'Doe', '-ibm -mail', 'janedoe@ acme.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createUser {-interactive}
```

- Using Jython string:

```
AdminTask.createUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createUser (['-interactive'])
```

deleteGroup

The **deleteGroup** command deletes a group in the virtual member manager. You cannot use this command to delete descendants. When this command completes, the group will be deleted from the repository.

Parameters and return values

-uniqueName
Specifies the unique name value for the group that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteGroup {-uniqueName cn=opera tors,cn=users,dc=yourco, dc=com}
```

- Using Jython string:

```
AdminTask.deleteGroup ('[-uniqueName cn=ope rators,cn=users,dc=you rco,dc=com]')
```

- Using Jython list:

```
AdminTask.deleteGroup (['-uniqueName', 'cn =operators,cn=users,dc =yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteGroup {-interactive}
```

- Using Jython string:

```
AdminTask.deleteGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteGroup (['-interactive'])
```

deleteUser

The **deleteUser** command deletes a user from the virtual member manager. This includes a person object and an account object in the non-merged repositories.

Parameters and return values

-uniqueName
Specifies the unique name value for the user that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteUser {-uniqueName uid=dmey ers,cn=users,dc=yourco, dc=com}
```

- Using Jython string:

```
AdminTask.deleteUser ('[-uniqueName uid= dmeyers,cn=users,dc= yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.deleteUser ([' -uniqueName', 'uid=dm eyers,cn=users,dc=yourco, dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteUser {-interactive}
```

- Using Jython string:

```
AdminTask.deleteUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteUser (['-interactive'])
```

duplicateMembershipOfGroup

Use the **duplicate Membership OfGroup** command to make a one group a member of all of the same groups as another group. For example, group A is in group B and group C. To add group D to the same groups as group A, use the **duplicate Membership OfGroup** command.

Parameters and return values

-copyToUniqueName
Specifies the name of the group to which you want to add the memberships of the group specified in the copyFromUniqueName parameter. (String, required)

-copyFromUniqueName
Specifies the name of the group from which you want to copy the group memberships for another group to use. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask duplicateMember shipOfGroup {-copyToUniqueName cn=operators,cn=groups, dc=yourco,dc=com
-copyFromUniqueName cn=admins,cn= groups,dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.duplicateMembershipOfGroup ('[-copyToUniqueName cn=operators,cn=groups, dc=yourco,dc=com
-copyFromUniqueName cn=admins,cn=gr oups,dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.duplicateMembershipOfGroup (['-copyToUniqueName', 'cn=operators,cn=groups, dc=yourco,dc=com',
'-copyFromUniqueName', 'cn=admins,cn =groups,dc=yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask duplicateMember shipOfGroup {-interactive}
```

- Using Jython string:

```
AdminTask.duplicateMember shipOfGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.duplicateMember shipOfGroup (['-interactive'])
```

duplicateMembershipOfUser

Use the **duplicate Membership OfUser** command to make a one user a member of all of the same groups as another user. For example, user 1 is in group B and group C. To add user 2 to the same groups as user 1, use the **duplicate Membership OfUser** command.

Parameters and return values

-copyToUniqueName
Specifies the name of the user to which you want to add the memberships of the user specified in the copyFromUniqueName parameter. (String, required)

-copyFromUniqueName
Specifies the name of the user from which you want to copy the group memberships for another user to use. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask duplicateMembershipOfUser {-copyToUniqueName uid=meyersd,cn=users,dc =yourco,dc=com
-copyFromUniqueName uid=jhart,cn= users,dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.duplicateMembershipOfUser ('[-copyToUniqueName uid=meyersd,cn=users,d c=yourco,dc=com
-copyFromUniqueName uid=jhart,cn =users,dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.duplicateMembershipOfUser (['-copyToUniqueName', 'uid=meyersd,cn=users, dc=yourco,dc=com',
'-copyFromUniqueName', 'uid= jhart,cn=users,dc=yourco, dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask duplicateMember shipOfUser {-interactive}
```

- Using Jython string:

```
AdminTask.duplicateMember shipOfUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.duplicateMember shipOfUser (['-interactive'])
```

getGroup

The **getGroup** command retrieves the common name and description of a group.

Parameters and return values

-uniqueName
Specifies the unique name value for the group that you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getGroup {-uniqueName cn=operators, cn=groups,dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.getGroup ('[ -uniqueName cn=operators, cn=groups,dc=yourco, dc=com]')
```

- Using Jython list:

```
AdminTask.getGroup ([' -uniqueName', 'cn=opera tors,cn=groups,dc=your co,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getGroup {-interactive}
```

- Using Jython string:

```
AdminTask.getGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getGroup (['-interactive'])
```

getMembershipOfGroup

The **getMembership OfGroup** command retrieves the groups of which a group is a member.

Parameters and return values

-uniqueName
Specifies the unique name value for the group whose group memberships you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getMebmership OfGroup {-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.getMebmership OfGroup ('[-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.getMebmership OfGroup (['-uniqueName', 'uid=dmeyers,cn=users, dc=yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getMembership OfGroup {-interactive}
```

- Using Jython string:

```
AdminTask.getMembership OfGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getMembership OfGroup (['-interactive'])
```

getMembershipOfUser

The **getMem bership OfUser** command retrieves the groups of which a user is a member.

Parameters and return values

-uniqueName
Specifies the unique name value for the user whose group memberships you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getMebmer shipOfUser {-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.getMebmership OfUser ('[-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.getMebmership OfUser (['-uniqueName', 'uid=dmeyers,cn=users ,dc=yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getMembership OfUser {-interactive}
```

- Using Jython string:

```
AdminTask.getMembership OfUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getMembership OfUser (['-interactive'])
```

getMembersOfGroup

The **getMembers OfGroup** command retrieves the members of a group.

Parameters and return values

-uniqueName
Specifies the unique name value for the group whose members you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getMembersOf Group {-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.getMembersOf Group ['(-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.getMembersOf Group [('-uniqueName', 'cn=operators,cn=groups ,dc=yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getMembersOfGroup {-interactive}
```

- Using Jython string:

```
AdminTask.getMembersOfGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getMembersOfGroup (['-interactive'])
```

getUser

The **getUser** command retrieves information about a user in the virtual member manager.

Parameters and return values

-uniqueName
Specifies the unique name value for the user that you want to view. This parameter maps to the uniqueName property in the virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask getUser {-user Name uid=dmeyers,cn=us ers,dc=yourco,dc=com}
```

- Using Jython string:

```
AdminTask.getUser ('[-use rName uid=dmeyers,cn= users,dc=yourco,dc=com]')
```

- Using Jython list:

```
AdminTask.getUser (['-use rName', 'uid=dmeyers,c n=users,dc=yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getUser {-interactive}
```

- Using Jython string:

```
AdminTask.getUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getUser (['-interactive'])
```

removeMemberFromGroup

The **removeMember FromGroup** command removes a user or a group from a group.

Parameters and return values

-memberUniqueName
Specifies the unique name value for the user or group that you want to remove from the specified group. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-groupUniqueName
Specifies the unique name value for the group from which you want to remove the user or group that you specified with the memberUniqueName paramter. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeMember FromGroup {-memberUnique Name uid=meyersd,cn= users,dc=yourco,dc=com
-groupUniqueName cn= admins,cn-groups,dc= yourco,dc=com}
```

- Using Jython string:

```
AdminTask.removeMemberF romGroup ('[-memberUnique Name uid=meyersd,cn= users,dc=yourco,dc=com
-groupUniqueName cn=a dmins,cn-groups,dc=your co,dc=com]')
```

- Using Jython list:

```
AdminTask.removeMemberFrom Group (['-memberUniqueName', 'uid=meyersd,cn=users, dc=yourco,dc=com',
'-groupUniqueName', 'cn=admins,cn-groups,dc= yourco,dc=com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeMember FromGroup {-interactive}
```

- Using Jython string:

```
AdminTask.removeMemberFr omGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeMemberFr omGroup (['-interactive'])
```

searchGroups

Use the **searchGroups** command to find groups in the virtual member manager that match criteria that you provide. For example, you can use the **searchGroups** command to find all of the groups with a common name that begins with `IBM`. You can search for any virtual member manager property because the command is generic.

Parameters and return values

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cn
The first name or given name of the user. This parameter maps to the cn property in the virtual member manager. You must set this parameter or the description parameter, but not both. (String, optional)

-description
Specifies information about the group. This parameter maps to the description entity in a virtual member manager object. You must set this parameter or the cn parameter, but not both. (String, optional)

-timeLimit
Specifies the maximum amount of time in milliseconds that the search can run. The default value is no time limit. (String, optional)

-countLimit
Specifies the maximum number of results that you want returned from the search. By default, all groups found in the search are returned. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask searchGroups {-cn *IBM*}
```

- Using Jython string:

```
AdminTask.searchGroups ('[-cn *IBM*]')
```

- Using Jython list:

```
AdminTask.searchGroups (['-cn', '*IBM*'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask searchGroups {-interactive}
```

- Using Jython string:

```
AdminTask.searchGroups ('[-interactive]')
```

- Using Jython list:

```
AdminTask.searchGroups (['-interactive'])
```

searchUsers

Use the **searchUsers** command to find users in the virtual member manager that match criteria that you provide. For example, you can use the **searchUsers** command to find all of the telephone numbers that contain 919. You can search for any virtual member manager property because the command is generic.

Parameters and return values

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-principalName
Specifies the principal name oft he user that is used as the logon ID for the user in the system. This parameter maps to the principalName property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)

-uid
Specifies the unique ID value for the user for whom you want to search. This parameter maps to the uid property in virtual member manage. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)

-cn
Specifies the first name or given name of the user. This parameter maps to the cn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)

-sn
Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)

-ibm-primaryEmail
Specifies the email address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)

-timeLimit
Specifies the maximum amount of time in milliseconds that the search can run. The default is not time limit. (String, optional)

-countLimit
Specifies the maximum number of results that you want returned from the search. By default, all users found int he search are returned. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask searchUsers {-principalName */IBM/US*}
```

- Using Jython string:

```
AdminTask.searchUsers (' [-principalName */IBM/US*]')
```

- Using Jython list:

```
AdminTask.searchUsers (['-principalName', '*/IBM/US*'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask searchUsers {-interactive}
```

- Using Jython string:

```
AdminTask.searchUsers ('[-interactive]')
```

- Using Jython list:

```
AdminTask.searchUsers (['-interactive'])
```

updateGroup

The **updateGroup** command updates the common name or the description of a group.

Parameters and return values

-uniqueName
Specifies the unique name value for the group for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-cn
Specifies the new common name used for the group. This parameter maps to the cn property in virtual member manager. (String, optional)

-description
Specifies the new information about the group. This parameter maps to the description entity in a virtual member manager object. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateGroup {-uniqueName cn=opera tors,cn=groups,dc=yourco ,dc=com -cn groupA}
```

- Using Jython string:

```
AdminTask.updateGroup ('[-uniqueName cn=oper ators,cn=groups,dc=your co,dc=com -cn groupA]')
```

- Using Jython list:

```
AdminTask.updateGroup ([' -uniqueName', 'cn=oper ators,cn=groups,dc=yourco, dc=com', '-cn', 'groupA'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateGroup {-interactive}
```

- Using Jython string:

```
AdminTask.updateGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateGroup (['-interactive'])
```

updateUser

The **updateUser** command updates the following properties: uniqueName, uid, password, cn, sn, or ibm-primaryEmail.

Parameters and return values

-uniqueName
Specifies the unique name value for the user for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required)

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-uid
Specifies the new unique ID value for the user. This parameter maps to the uid property in virtual member manager. (String, optional)

-password
Specifies the new password for the user. This parameter maps to the password property in virtual member manager. (String, optional)

-confirmPassword
Specifies the password again to validate how it was entered on the password parameter. This parameter maps to the password property in virtual member manager. (String, optional)

-cn
Specifies the new first name or given name of the user. This parameter maps to the cn property in virtual member manager. (String, optional)

-surname
Specifies the new last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, optional)

-ibm-primaryEmail
Specifies the new email address of the user. This parameter maps to the mail property in virtual member manager. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask updateUser {-uniqueName uid=dmey ers,cn=users,dc=yourco, dc=com -uid 123}
```

- Using Jython string:

```
AdminTask.updateUser ('[-uniqueName uid= dmeyers,cn=users,dc= yourco,dc=com -uid 123]')
```

- Using Jython list:

```
AdminTask.updateUser (['-uniqueName', 'uid =dmeyers,cn=users,dc= yourco,dc=com', '-uid', '123'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask updateUser {-interactive}
```

- Using Jython string:

```
AdminTask.updateUser ('[-interactive]')
```

- Using Jython list:

```
AdminTask.updateUser (['-interactive'])
```

DescriptivePropCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the DescriptivePropCommands group can be used to create, delete, and manage key manager setting in your configuration.

The DescriptivePropCommands command group for the AdminTask object includes the following commands:

- “deleteDescriptiveProp”
- “getDescriptiveProp” on page 493
- “listDescriptiveProp” on page 493
- “modifyDescriptiveProp” on page 494

deleteDescriptiveProp

The **deleteDescriptiveProp** command deletes key manager settings from the configuration.

Target object

None

Parameters and return values

-parentDataType
(String, required)

-parentClassName
(String, required)

-parentScopeName
(String, optional)

-name
(String, required)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteDescrip tiveProp {-interactive}
```

- Using Jython string:

```
AdminTask.deleteDescrip tiveProp ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteDescrip tiveProp (['-interactive'])
```

getDescriptiveProp

The **getDescriptiveProp** command obtains information about key manager settings.

Target object

None

Parameters and return values

-parentDataType
(String, required)

-parentClassName
(String, required)

-parentScopeName
(String, optional)

-name
(String, required)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask getDescripti veProp {-interactive}
```

- Using Jython string:

```
AdminTask.getDescriptiv eProp ('[-interactive]')
```

- Using Jython list:

```
AdminTask.getDescriptive Prop (['-interactive'])
```

listDescriptiveProp

The **listDescriptiveProp** command lists the key managers within a particular management scope.

Target object

None

Parameters and return values

-parentDataType
(String, required)

-parentClassName
(String, required)

-parentScopeName
(String, optional)

-displayObjectName
Set the value of this parameter to `true` to list the key manager objects within the scope. Set the value of this parameter to `false` to list the strings that contain the key manager name and management scope. (Boolean, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listDescrip tiveProp {-interactive}
```

- Using Jython string:

```
AdminTask.listDescrip tiveProp ('[-interact ive]')
```

- Using Jython list:

```
AdminTask.listDescrip tiveProp (['-interact ive'])
```

modifyDescriptiveProp

The **modifyDescriptiveProp** command modifies the settings of an existing key manager.

Target object

None

Parameters and return values

-parentDataType
(String, required)

-parentClassName
(String, required)

-parentScopeName
(String, optional)

-name
(String, required)

-value
(String, optional)

-type
(String, optional)

-displayNameKey
(String, optional)

-nlsRangeKey
(String, optional)

-hoverHelpKey
(String, optional)

-range
(String, optional)

-inclusive
(Boolean, optional)

-firstClass
(Boolean, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyDescrip tiveProp {-interactive}
```

- Using Jython string:

```
AdminTask.modifyDescrip tiveProp ('[-interacti ve]')
```

- Using Jython list:

```
AdminTask.modifyDescrip tiveProp (['-interactiv e'])
```

ManagementScopeCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. Inbound and outbound management scopes represent opposing directions during the connection handshake process. The commands and parameters in the ManagementScopeCommands group can be used to create, delete, and list management scopes.

The ManagementScopeCommands command group for the AdminTask object includes the following commands:

- "deleteManagementScope"
- "getManagementScope" on page 496
- "listManagementScopes" on page 496

deleteManagementScope

The **deleteManage mentScope** command deletes a management object from the configuration.

Target object

None

Parameters and return values

- scopeName
The name that uniquely identifies the management scope. (String, required)

Examples

Batch mode example usage:

Interactive mode example usage:

getManagementScope

The **getManagementScope** command displays the setting of a management scope object.

Target object

None

Parameters and return values

- scopeName
The name that uniquely identifies the management scope. (String, required)

Examples

Batch mode example usage:

Interactive mode example usage:

listManagementScopes

The **listManagementScopes** command lists the management scopes in the configuration.

Target object

None

Parameters and return values

- displayObjectName
Set the value to true to display the object names of the management scope. (Boolean, optional)

Examples

Batch mode example usage:

Interactive mode example usage:

AuthorizationGroupCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the AuthorizationGroupCommands group can be used to create and manage authorization groups.

The AuthorizationGroupCommands command group for the AdminTask object includes the following commands:

addResourceToAuthorizationGroup

The **addResourceToAuthorizationGroup** command adds a resource instance to an existing authorization group. A resource instance cannot belong to more than one authorization group.

Target object

None

Parameters and return values

- authorizationGroupName
The name of the authorization group. (String, required)

- resourceName
The name of the resource instance that you want to add to an authorization group. (String, required)

The resourceName parameter should be in the following format:

```
ResourceType=ResourceName
```

where `ResourceType` is one of the following values: Application, Server, ServerCluster, Node, NodeGroup

`ResourceName` is the name of the resource instance, for example, `server1`.

The following are example uses of the resourceName parameter:

-
```
Node=node1:Server=server1
```
This example uniquely identifies `server1`. `node1` is required if another `server1` exists on a different node.
-
```
Application=app1
```

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addResourceToAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}
```

- Using Jython string:

```
AdminTask.addResourceToAuthorizationGroup('[-authorizationGroupName groupName -resourceName Application=app1]')
```

- Using Jython list:

```
AdminTask.addResourceToAuthorizationGroup(['-authorizationGroupName', 'groupName', '-resourceName', 'Application=app1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addResourceToAuthorizationGroup {-interactive}
```

- Using Jython string:

```
AdminTask.addResourceToAuthorizationGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addResourceToAuthorizationGroup (['-interactive'])
```

createAuthorizationGroup

The **createAuthorizationGroup** command creates a new authoirzation group. When you create a new authorization group, no members are associated with it. Also, no user to administrative role mapping for the authorization table is associated with the authorization group.

Target object

None

Parameters and return values

- authorization GroupName
The name of the authorization group that you want to create. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createAuthorizationGroup {-authorizationGroupName groupName}
```

- Using Jython string:

```
AdminTask.createAuthorizationGroup('[-authorizationGroupName groupName]')
```

- Using Jython list:

```
AdminTask.createAuthorizationGroup(['-authorizationGroupName', 'groupName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createAuthorizationGroup -interactive
```

- Using Jython string:

```
AdminTask.createAuthorizationGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createAuthorizationGroup (['-interactive'])
```

deleteAuthorizationGroup

The **deleteAuthorizationGroup** command deletes an existing authorization group. When you delete an authorization group, the authorization table that corresponds is also deleted.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group that you want to delete. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}
```

- Using Jython string:

```
AdminTask.deleteAuthorizationGroup('[-authorizationGroupName groupName]')
```

- Using Jython list:

```
AdminTask.deleteAuthorizationGroup(['-authorizationGroupName', 'groupName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteAuthorizationGroup  {-interactive}
```

- Using Jython string:

```
AdminTask.deleteAuthorizationGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteAuthorizationGroup (['-interactive'])
```

listAuthorizationGroups

The **listAuthorizationGroupslistAuthoriz ationGroups** command lists the existing authorization groups.

Target object

None

Parameters and return values

- Parameters: None
- Returns: A list of short names of all existing authorization groups. (String [])

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroups
```

- Using Jython:

```
AdminTask.listAuthorizationGroups()
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroups {-interactive}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroups ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroups (['-interactive'])
```

listAuthorizationGroupsForGroupID

The **listAuthorizationGroupsForGroupID** command lists all of the authorization groups to which a given user group has access. This command lists the authorization groups and the granted roles for each authorization group. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list `cell` as a group if the user has cell level access.

Target object

None

Parameters and return values

- groupid
The ID of the user group. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsForGroupID {-groupid userGroupName}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsForGroupID('[-groupid userGroupName]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsForGroupID(['-groupid', 'userGroupName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsForGroupID {-interactive}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsForGroupID ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsForGroupID (['-interactive'])
```

listAuthorizationGroupsForUserID

The **listAuthorizationGroupsForUserID** command lists all of the authorization groups to which a given user has access. This command lists the authorization groups and the granted roles for each authorization group. The user ID and the group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list `cell` as a group if the user has cell level access.

Target object

None

Parameters and return values

- userid
The ID of the user. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsForUserID{-userid userName}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsForUserID('[-userid userName]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsForUserID(['-userid', 'userName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsForUserID {-interactive}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsForUserID ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsForUserID (['-interactive'])
```

listAuthorizationGroupsOfResource

The **listAuthorizationGroupsOfResource** command lists authorization groups for a given resource. If the value of the traverseContainedObjects parameter is false, only the authorization group of the resource is returned. If the value of the traverseContainedObjects parameter is true, it returns the authorization group of the resource and the authorization groups of all the parent resources in the containment tree.

Target object

None

Parameters and return values

- resourceName
The name of the resource. (String, required)

The resourceName parameter must be in the following format:

```
ResourceType=ResourceName
```

where `ResourceType` can be any one of the following values: Application, Server, ServerCluster, Node, or NodeGroup.

`ResourceName` is the name of the resource instance, for example, server1.

The following are examples of the resourceName parameter:

```
Node=node1:Server=server
```

This example uniquely identifies *server1*. The name of the node is required if a server on a different node uses the same server name.

```
Application=app1
```

- traverseContained Resources
Finds the authorization groups of all the parent resources by traversing the resource containment tree upwards. The default value is false. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsOfResource {-resourceName Application=app1}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsOfResource('[-resourceName Application=app1]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsOfResource(['-resourceName', 'Application=app1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAuthorizationGroupsOfResource {-interactive}
```

- Using Jython string:

```
AdminTask.listAuthorizationGroupsOfResource ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listAuthorizationGroupsOfResource (['-interactive'])
```

listResourcesOfAuthorizationGroup

The **listResourcesOfAuthorizationGroup** command lists all of the resources within the given authorization group.

Target object

None

Parameters and return values

- authorizationGroupName
The name of the authorization group. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listResourcesOfAuthorizationGroup {-authorizationGroupName groupName}
```

- Using Jython string:

```
AdminTask.listResourcesOfAuthorizationGroup('[-authorizationGroupName groupName]')
```

- Using Jython list:

```
AdminTask.listResourcesOfAuthorizationGroup(['-authorizationGroupName', 'groupName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listResourcesOfAuthorizationGroup {-interactive}
```

- Using Jython string:

```
AdminTask.listResourcesOfAuthorizationGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listResourcesOfAuthorizationGroup (['-interactive'])
```

listResourcesForGroupID

The **listResourcesForGroupID** command lists all the objects that a given group has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user group is granted roles and the resources that are descendants of the resources with in authorization groups to which the user group is granted access to any role. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.

Target object

None

Parameters and return values

- groupid
The ID of the user group. (String, required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listResourcesForGroupID {-groupid userGroupName}
```

- Using Jython string:

```
AdminTask.listResourcesForGroupID('[-groupid userGroupName]')
```

- Using Jython list:

```
AdminTask.listResourcesForGroupID(['-groupid', 'userGroupName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listResourcesForGroupID {-interactive}
```

- Using Jython string:

```
AdminTask.listResourcesForGroupID ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listResourcesForGroupID (['-interactive'])
```

listResourcesForUserID

The **listResourcesForUserID** command lists all the objects that a given user has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user is granted roles and the resources that are descendants of the resources with in authorization groups to which the user is granted access to any role. The user ID can be a short name or fully qualified domain name if a LDAP user registry is used.

Target object

None

Parameters and return values

- userid
The ID of the user. (String, required).

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listResourcesForUserID {-userid userName }
```

- Using Jython string:

```
AdminTask.listResourcesForUserID('[-userid userName]')
```

- Using Jython list:

```
AdminTask.listResourcesForUserID(['-userid', 'userName'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listResourcesForUserID {-interactive}
```

- Using Jython string:

```
AdminTask.listResourcesForUserID ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listResourcesForUserID (['-interactive'])
```

Example output:

```
{deployer=[], operator=[],  administrator=[cells/IBM-LP1 6L31HVE8Cell07/clusters/C1| cluster.xml,
 cells/IBM-LP16L 31HVE8Cell07/nodes/IBM-LP16L 31HVE8Node05/servers/cm1|ser ver.xml],
 monitor=[], configurator=[]}
```

mapGroupsToAdminRole

The **mapGroupsToAdminRole** command maps group IDs to one or more administrative roles in an authorization group. The name of the authorization group that you provide determines which authorization table will be used. If you do not specify an authorization group name, the mapping is done to the cell level authorization table. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is used.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group. If you do not specify this parameters, the cell level authorization group is assumed. (String, optional)

- roleName
The name of the administrative role. (String, required)

- groupids
The list of group IDs that will mapped to the administrative role. (String[], required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask mapGroupsToAdminRole {-authorizationGroupName groupName - roleName administrator -groupids group1}
```

- Using Jython string:

```
AdminTask.mapGroupsToAdminRole('[-authorizationGroupName groupName -roleName administrator -groupids group1]')
```

- Using Jython list:

```
AdminTask.mapGroupsToAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator',
'-groupids', 'group1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask mapGroupsToAdminRole {-interactive}
```

- Using Jython string:

```
AdminTask.mapGroupsToAdminRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.mapGroupsToAdminRole (['-interactive'])
```

mapUsersToAdminRole

The **mapUsersToAdminRole** command maps user IDs to one or more administrative roles in the authorization group. The name of the authorization group that you provide determines the authorization table. If you do not specify the name of the authorization group, the mapping is done to the cell level authorization table. The user ID can be a short name or fully qualified domain name in case LDAP user registry is used.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional)

- roleName
The name of the administrative role. (String, required)

- userids
The list of user IDs that will be mapped to the administrative role (String[], required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask mapUsersToAdminRole {-authorizationGroupName groupName - roleName administrator -userids user1}
```

- Using Jython string:

```
AdminTask.mapUsersToAdminRole('[-authorizationGroupName groupName -roleName administrator -userids user1]')
```

- Using Jython list:

```
AdminTask.mapUsersToAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator',
'-userids', 'user1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask mapUsersToAdminRole {-interactive}
```

- Using Jython string:

```
AdminTask.mapUsersToAdminRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.mapUsersToAdminRole (['-interactive'])
```

removeGroupsFromAdminRole

The **removeGroupsFromAdminRole** command removes previously mapped group IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the group IDs are removed from the cell level authorization table. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional)

- roleName
The name of the administrative role. (String, required)

- userids
A list of group IDs that you want to remove from the administrative role. (String[], required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeGroupsFromAdminRole {-authorizationGroupName groupName - roleName administrator -groupids group1}
```

- Using Jython string:

```
AdminTask.removeGroupsFromAdminRole('[-authorizationGroupName groupName -roleName administrator -groupids group1]')
```

- Using Jython list:

```
AdminTask.removeGroupsFromAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator',
 '-groupids', 'group1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeGroupsFromAdminRole {-interactive}
```

- Using Jython string:

```
AdminTask.removeGroupsFromAdminRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeGroupsFromAdminRole (['-interactive'])
```

removeResourceFromAuthorizationGroup

The **removeResourceFromAuthorizationGroup** command removes resources from an existing authorization group. If you do not specify the authorization group, it will be determined and the resource will be removed from that authorization group.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group. (String, optional)

- resourceName
The name of the resource instance that you want to remove from the authorization group. (String, required)

The resourceName parameter must be in the following format:

```
ResourceType=ResourceName
```

where the `ResourceType` can be any of the following: Application, Server, ServerCluster, Node, or NodeGroup.

The `ResourceName` is the name of the resource instance, for example, server1.

The following are examples of the resourceName parameter:

```
Node=node1:Server=server1
```

This example uniquely identifies server1. node1 is required if the name of the server exists on multiple nodes.

```
Application=app1
```

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeResourceFromAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}
```

- Using Jython string:

```
AdminTask.removeResourceFromAuthorizationGroup('[-authorizationGroupName groupName -resourceName Application=app1]')
```

- Using Jython list:

```
AdminTask.removeResourceFromAuthorizationGroup(['-authorizationGroupName',  'groupName', '-resourceName',
 'Application=app1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeResourceFromAuthorizationGroup {-interactive}
```

- Using Jython string:

```
AdminTask.removeResourceFromAuthorizationGroup ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeResourceFromAuthorizationGroup (['-interactive'])
```

removeUsersFromAdminRole

The **removeUsersFromAdminRole** command removes previously mapped user IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the user ID from the cell level authorization table will be used. The user ID can be a short name or a fully qualified domain name if a LDAP user registry is used.

Target object

None

Parameters and return values

- authorizationGroup Name
The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional)

- roleName
The name of the administrative role. (String, required)

- userids
A list of user IDs that you want to remove from the administrative role. (String[], required)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask removeUsersFromAdminRole {-authorizationGroupName groupName - roleName administrator -userids user1}
```

- Using Jython string:

```
AdminTask.removeUsersFromAdminRole('[-authorizationGroupName groupName -roleName administrator -userids user1]')
```

- Using Jython list:

```
AdminTask.removeUsersFromAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator',
'-userids', 'user1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask removeUsersFromAdminRole {-interactive}
```

- Using Jython string:

```
AdminTask.removeUsersFromAdminRole ('[-interactive]')
```

- Using Jython list:

```
AdminTask.removeUsersFromAdminRole (['-interactive'])
```

ChannelFrameworkManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security. The commands and parameters in the ChannelFrameworkManagement group can be used to create and manage transport channels and transport channel chains.

The ChannelFrameworkManagement command group for the AdminTask object includes the following commands:

createChain

The **createChain** command creates a new chain of transport channels that are based on a chain template.

Target object

The instance of the transport channel service under which the new chain is created. (ObjectName, required)

Required parameters and return values

- template
The chain template on which to base the new chain. (ObjectName, required)

- name
The name of the new chain. (String, required)

- endPoint
The name of the end point to be used by the instance of the TCP inbound channel in the new chain if the chain is an inbound chain. (ObjectName, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createChain (cells/ rohitbuildCell01/nodes/rohit buildCellManager01/servers/
 dmgr|server.xml#TransportCha  nnelService_1) {-template  WebContainer(templates/ chains|webcontainer-chains.xml #Chain_1)
 -name trialChain1}

$AdminTask createChain (cells/ rohitbuildCell01/nodes/rohitb uildCellManager01/servers/dmgr
 |server.xml#TransportChannel  Service_1) {-template  WebContainer(templates/ chains|webcontainer-chains.xml# Chain_1)
 -name trialChain1   -endPoint (cells/rohitbuild Cell01/nodes/rohitbuildCellMa nager01|serverindex.xml#End Point_3) }
```

- Using Jython string:

```
  AdminTask.createChain('cells/ rohitbuildCell01/nodes/rohitbu ildCellManager01/servers/dmgr|
 server.xml#TransportChannelSer  vice_1', '[-template "WebConta iner(templates/chains|webconta iner-chains.xml#Chain_1)" -name
 trialChain]')

AdminTask.createChain('cells/ rohitbuildCell01/nodes/rohit buildCellManager01/servers/dmgr
 |server.xml#TransportChannel Service_1',  '[-template "WebCo ntainer(templates/chains|webc ontainer-chains.xml#Chain_1)"  -name
 trialChain -endPoint   "(cells/rohitbuildCell01/nodes/ rohitbuildCellManager01|server index.xml#EndPoint_3)"]')
```

- Using Jython list:

```
AdminTask.createChain('cells/ rohitbuildCell01/nodes/ rohitbuildCellManager01/serve
 rs/dmgr|server.xml# TransportChannelService_1',   ['-template', "WebContainer (templates/chains|webcontaine
 r-chains.xml#Chain_1)",  '-name', 'trialChain'])

AdminTask.createChain('cells/ rohitbuildCell01/nodes/rohit buildCellManager01/servers/
 dmgr|server.xml#TransportCha nnelService_1',  ['-template',  "WebContainer(templates/chains |webcontainer-chains.xml#Cha in_1)",
 '-name', 'trialChain',  '-endPoint',  "(cells/rohitbu ildCell01/nodes/rohitbuildCe llManager01|serverindex.xml#
 EndPoint_3)"])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createChain {-interactive}
```

- Using Jython string:

```
AdminTask.createChain ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createChain (['-interactive'])
```

deleteChain

The **deleteChain** command deletes an existing chain and, optionally, the transport channels in the chain.

Target object

The chain to be deleted. (Object name, required)

Required parameters and return values

- deleteChannels
If the value of this attribute is true, non-shared transport channels used by the specified chain will be deleted. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteChain trialChain1 (cells/rohitbuildCell01/nodes/roh
 itbuildCellManager01/servers/dmgr | server.xml#Chain_1093554462922)
$AdminTask deleteChain trialChain (cells/rohitbuildCell01/nodes/roh
 itbuildCellManager01/servers/dmgr  |server.xml#Chain_1093554378078)  {-deleteChannels true}
```

- Using Jython string:

```
  AdminTask.deleteChain('trialChain 1(cells/rohitbuildCell01/nodes/roh
 itbuildCellManager01/servers/dmgr|  server.xml#TransportChannelSer vice_1)')
AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/rohi
 tbuildCellManager01/servers/dmgr|  server.xml#TransportChannelService _1)', '[-deleteChannels true]')
```

- Using Jython list:

```
AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/roh
 itbuildCellManager01/servers/dmgr|  server.xml#TransportChannelService _1)')
AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/rohi
 tbuildCellManager01/servers/dmgr|  server.xml#TransportChannelServic e_1)', ['-deleteChannels', 'true'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteChain {-interactive}
```

- Using Jython string:

```
AdminTask.deleteChain ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteChain (['-interactive'])
```

listChainTemplates

The **listChain Templates** command displays a list of templates that you can use to create chains in this configuration. All templates have a certain type of transport channel as the last transport channel in the chain.

Target object

None

Required parameters and return values

- acceptorFilter
The templates returned by this method all have a transport channel instance of the specified type as the last transport channel in the chain. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listChainTemplates {}
$AdminTask listChainTemplates  "-acceptorFilter WebContainer InboundChannel"
```

- Using Jython string:

```
AdminTask.listChainTemplates()
AdminTask.listChainTemplates ('[-acceptorFilter WebCont ainerInboundChannel]')
```

- Using Jython list:

```
AdminTask.listChainTemplates()
AdminTask.listChainTemplates (['-acceptorFilter', 'WebC ontainerInboundChannel'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listChainTemplates {-interactive}
```

- Using Jython string:

```
AdminTask.listChainTemplates ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listChainTemplates (['-interactive'])
```

listChains

The **listChains** command lists all the chains that are configured under a particular instance of the transport channel service.

Target object

The instance of the transport channel service under which the chains are configured. (ObjectName, required)

Required parameters and return values

- acceptorFilter
The chains that are returned by this parameter will have a transport channel instance of the type that you specify as the last transport channel in the chain. (String, optional)

- endPointFilter:
The chains returned by this parameter will have a TCP inbound channel using an end point with the name that you specify.(String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listChains (cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)
$AdminTask listChains (cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)  {-acceptorFilter WebConta inerInboundChannel}
```

```
$AdminTask listChains (cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328) {-end PointFilter WC_adminhost}
```

- Using Jython string:

```
AdminTask.listChains('(cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)')

AdminTask.listChains('(cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)',  '[-acceptorFilter WebContai nerInboundChannel]')

AdminTask.listChains('(cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)',  '[-endPointFilter WC_adminhost]')
```

- Using Jython list:

```
AdminTask.listChains('(cells/ rohitbuildCell01/nodes/rohit buildNode01/servers/server2|
 server.xml#TransportChannel Service_1093445762328)')

AdminTask.listChains('(cells /rohitbuildCell01/nodes/roh itbuildNode01/servers/server 2
  |server.xml#TransportChanne lService_1093445762328)',  ['-acceptorFilter', 'WebCon tainerInboundChannel'])

AdminTask.listChains('(cells /rohitbuildCell01/nodes/roh itbuildNode01/servers/server
 2|server.xml #TransportChanne lService_1093445762328)',  ['-endPointFilter', 'WC_admi nhost'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listChains {-interactive}
```

- Using Jython string:

```
AdminTask.listChains ('[-interactive]')
```

- Using Jython list:

```
AdminTask.listChains (['-interactive'])
```

SpnegoTAICommands group for the AdminTask object (deprecated)

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the SpnegoTAICommands group can be used to create and manage configurations that are used by the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI).

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server Version 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

The SpnegoTAICommands command group for the AdminTask object includes the following commands:

- “addSpnegoTAIProperties”
- “deleteSpnegoTAIProperties” on page 513
- “modifySpnegoTAIProperties” on page 513
- “showSpnegoTAIProperties” on page 514
- “createKrbConfigFile” on page 515

addSpnegoTAIProperties

The **addSpnego TAI Properties** command adds properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for the application server.

Target object

None

Parameters and return values

-spnId
This is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned. (String, optional)

-host
Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, required)

-filter
Defines the filtering criteria used by the class specified with the above attribute. If no filter is specified, all HTTP requests are subject to SPNEGO authentication. (String, optional)

-filterClass
Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no filter class is specified, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used. (String, optional)

-noSpnegoPage
Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional).

If you do not specify the noSpnegoPage attribute then the default is used:

```
"<html><head><title> SPNEGO authentication is not supported. </title></head>" +
"<body>SPNEGO authe ntication is not supported on this client.</body> </html>";
```

-ntlmTokenPage
Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional).

If you do not specify the ntlmTokenPage attribute then the default is used:

```
"<html><head><title> An NTLM Token was received.</title> </head>" + "<body>Your bro wser
configuration is correct, but you have not logged into a supported Windows Domain." +
"<p>Please login to the application using the normal login page.</html>";
```

-trimUserName
Specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the `@` that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask addSpnegoTAIProp erties -host myhost.ibm.com -filter user-agent%=IE 6
```

- Using Jython string:

```
AdminTask.addSpnegoTAIProp erties ('[-host myhost.ibm. com -filter user-agent%=IE 6]')
```

- Using Jython list:

```
AdminTask.addSpnegoTAIProp erties (['-host', 'myhost. ibm.com', '-filter', 'user -agent%=IE', '6'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask addSpnegoTAI Properties -interactive
```

- Using Jython string:

```
AdminTask.addSpnegoTAIPro perties ('[-interactive]')
```

- Using Jython list:

```
AdminTask.addSpnegoTAIPro perties ['-interactive'])
```

deleteSpnegoTAIProperties

The **deleteSpnego TAIProperties** command deletes properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Target object

None

Parameters and return values

-spnId
The SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask deleteSpnegoT AIProperties {-spnId 2}
```

- Using Jython string:

```
AdminTask.deleteSpnegoT AIProperties ('[-spnId 2]')
```

- Using Jython list:

```
AdminTask.deleteSpnegoTAI Properties (['-spnId', '2'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask deleteSpnegoTAI Properties -interactive
```

- Using Jython string:

```
AdminTask.deleteSpnegoTAI Properties ('[-interactive]')
```

- Using Jython list:

```
AdminTask.deleteSpnegoTAI Properties ['-interactive'])
```

modifySpnegoTAIProperties

The **modifySpnego TAIProperties** command modifies the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Target object

None

Parameters and return values

-spnId
The SPN identifier for the group of custom properties that are to be defined with this command. (String, required)

-host
Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, optional)

-filter
Defines the filtering criteria used by the class specified with the above attribute. (String, optional)

-filterClass
Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication. (String, optional)

-noSpnegoPage
Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional)

-ntlmTokenPage
Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional)

-trimUserName
Specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask modifySpnegoTAI PROPERTIES -spnId 1 -filter host==myhost.company.com
```

- Using Jython string:

```
AdminTask.modifySpnegoTAI PROPERTIES ('[-spnId 1 -filter host==myhost.com pany.com]')
```

- Using Jython list:

```
AdminTask.modifySpnegoTAI PROPERTIES (['-spnId', '1', '-filter', 'host==my host.company.com'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifySpnegoTAI Properties -interactive
```

- Using Jython string:

```
AdminTask.modifySpnegoTAI Properties ('[-interactive]')
```

- Using Jython list:

```
AdminTask.modifySpnegoTAI Properties ['-interactive'])
```

showSpnegoTAIProperties

The **showSpnego TAI Properties** command displays the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Target object

None

Parameters and return values

-spnId
The service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask showSpnegoTAI Properties -spnId 1
```

- Using Jython string:

```
AdminTask.showSpnegoTAI Properties ('[-spnId 1]')
```

- Using Jython list:

```
AdminTask.showSpnegoTAI Properties (['-spnId', '1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask showSpnegoTAI Properties -interactive
```

- Using Jython string:

```
AdminTask.showSpnegoTAI Properties ('[-interact ive]')
```

- Using Jython list:

```
AdminTask.showSpnegoTAI Properties ['-interact ive'])
```

createKrbConfigFile

The **createKrb ConfigFile** command creates the Kerberos configuration file for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Target object

None

Parameters and return values

-krbPath
Provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file. (String, required)

-realm
Provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property com.ibm.ws.security.spnego.SPN<id>.hostname (String, required)

-kdcHost
Provides the host name of the Kerberos Key Distribution Center (KDC). (String, required)

-kdcPort
Provides the port number of the KDC. The default value, if not specified, is 88. (String, optional)

-dns
Provides the default domain name service (DNS) that is used to produce a fully qualified host name. (String, required)

-keytabPath
Provides the file system location of the Kerberos keytab file. (String, required)

-encryption
Identifies the list of supported encryption types, separated by a space. The specified value is used for the default_tkt_enctypes and default_tgs_enctypes. The default encryption types, if not specified, are des-cbc-md5 and rc4-hmac. (String, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createKrbCo nfigFile -interactive
```

- Using Jython string:

```
AdminTask.createKrbCon figFile ('[-interactive]')
```

- Using Jython list:

```
AdminTask.createKrbCon figFile ['-interactive'])
```

The Kerberos configuration file

The Kerberos configuration properties, `krb5.ini` or `krb5.conf` files, must be configured on every WebSphere Application Server instance in a cell in order to use the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Note: In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

The default Kerberos configuration file name for Windows is krb5.ini. For other platforms is the default Kerberos configuration file name is krb5.conf. The default location for the Kerberos configuration file is shown below:

Table 61. Default locations for Kerberos configuration file. This table describes default locations for the Kerberos configuration file.

Operating System	Default Location
Windows	c:\winnt\krb5.ini **Note:** If the **krb5.ini** file is not located in the `c:\winnt` directory it might be located in `c:\windows` directory.
Linux	/etc/krb5.conf
other UNIX-based	/etc/krb5/krb5.conf
z/OS	/etc/krb5/krb5.conf
IBM i	/QIBM/UserData/OS400/NetworkAuthentication/krb5.conf

Note: If you do not use the default location and Kerberos configuration file name, then you have to update `*.krb5ConfigFile` properties in the soap.client.prop, ipc.client.props, and sas.client.props files. Also, if the client programmatic login uses the WSKRBLogin module, you must also set the `java.security.krb5.conf` JVM property.

For SPNEGO TAI, if you do not use the default location and Kerberos configuration file name, then you must specify the `java.security.krb5.conf` JVM property.

The default Kerberos configuration file on Windows is `/winnt/krb5.ini` and on a distributed environment is `/etc/krb5`. If you specify another location path, then you must also specify the `java.security.krb5.conf` JVM property.

For example, if your krb5.conf file is specified at /opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf, then you need to specify -Djava.security.krb5.conf=/opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf.

The WebSphere runtime code searches for the Kerberos configuration file in the order as follows:

1. The file referenced by the Java property java.security.krb5.conf
2. <java.home>/lib/security/krb5.conf
3. c:\winnt\krb5.ini on Microsoft Windows platforms
4. /etc/krb5/krb5.conf on UNIX platforms
5. /etc/krb5.conf on Linux platforms.

Use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory from the Qshell command line.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask createKrbConfigFile
   ```

   You can use the following parameters with this command:

Table 62. Command parameters. This table describes parameters for the $AdminTask createKrbConfigFile command.

Option	Description
<krbPath>	This parameter is required. It provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file.
<realm>	This parameter is required. It provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property com.ibm.ws.security.spnego.SPNid.hostName.
<kdcHost>	This parameter is required. It provides the host name of the Kerberos Key Distribution Center (KDC).
<kdcPort>	This parameter is optional. It provides the port number of the KDC. The default value, if not specified, is 88.
<dns>	This parameter is required. It provides the default domain name service (DNS) that is used to produce a fully qualified host name.
<keytabPath>	This parameter is required. It provides the file system location of the Kerberos keytab file.
<encryption>	This parameter is optional. It identifies the list of supported encryption types, separated by a space. The specified value is used for the default_tkt_enctypes and default_tgs_enctypes. The default encryption types, if not specified, are des-cbc-md5 and rc4-hmac.

In the following example, the wsadmin command creates the krb5.ini file in the c:\winnt directory. The default Kerberos keytab file is also in c:\winnt. The actual Kerberos realm name is WSSEC.AUSTIN.IBM.COM and the KDC host name is host1.austin.ibm.com.

```
wsadmin>$AdminTask createKrbConfigFile {-krbPath
c:\winnt\krb5.ini -realm WSSEC.AUSTIN.IBM.COM -kdcHost host1.austin.ibm.com
 -dns austin.ibm.com -keytabPath c:\winnt\krb5.keytab}
```

The wsadmin command above creates a krb5.ini file as follows:

```
[libdefaults]
 default_realm = WSSEC.AUSTIN.IBM.COM
        default_keytab_name = FILE:c:\winnt\krb5.keytab
        default_tkt_enctypes = des-cbc-md5 rc4-hmac
        default_tgs_enctypes = des-cbc-md5 rc4-hmac
[realms]
        WSSEC.AUSTIN.IBM.COM = {
  kdc = host1.austin.ibm.com:88
              default_domain = austin.ibm.com
}
[domain_realm]
        .austin.ibm.com = WSSEC.AUSTIN.IBM.COM
```

Attention:

- A Kerberos keytab file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk. The `krb5.conf` file permission must be 644, which means that you can read and write the file; however, members of the group that the file belongs to, and all others can only read the file.
- If the run time cannot read the `default_tkt_enctypes` or `default_tgs_enctypes` entries in the `krb5.ini` file, their values are missing, or their values are not supported, the `DES-CBC-MD5` value is used by default.

The krb5.conf configuration file supports trigraphs to represent the {, }, [, and] characters. These characters depend on the language set. The natively generated keytabs cannot be read by the Kerberos client. If you have difficulty configuring SPNEGO TAI with the native krb5.conf or krb5.keytab files, complete one of the following scenarios to address the trigraphs issue:

- Replace the trigraphs in the krb5.conf file with the characters that they represent.
- Use the krb5.conf file that is generated by WebSphere Application Server.
- Use a Microsoft Windows or a key distribution center (KDC) generated keytab file.

Kerberos configuration settings, the Kerberos key distribution center (KDC) name, and realm settings for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) are provided in the Kerberos configuration file or through `java.security.krb5.kdc` and `java.security.krb5.realm` system property files.

SPNEGO web authentication configuration commands

Use wsadmin commands to configure, unconfigure, validate, or display Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) in the security configuration.

Configure SPNEGO web authentication

Note: You must first have a workable Kerberos configuration file and a Kerberos keytab file. For more information, read topics about creating a Kerberos configuration file and creating a Kerberos service principal name and keytab file.

Use the configureSpnego command to configure SPNEGO as a web authenticator in the security configuration.

At the wsadmin prompt, enter the following command for help:

```
Wsadmin>$AdminTask help configureSpnego
```

Table 63. Command parameters. You can use the following parameters with the configureSpnego command:

Option	Description
<enabled>	This parameter is optional. It enables SPNEGO web authentication.
<dynamicReload>	This parameter is optional. It enables dynamic reload of SPNEGO web authentication filters.
<allowAppAuthMethodFallback>	This parameter is optional. It allows fall back to the application authentication mechanism.
<krb5Config>	This parameter is required. It supplies the directory location and file name of the configuration (krb5.ini or krb5.conf) file.
<krb5Keytab>	This parameter is optional. It supplies the directory location and file name of the Kerberos keytab file. If you do not specify this parameter, the default keytab in the Kerberos configuration file is used.

Note: WebSphere variables can be used to specify the krb5Config and krb5Keytab file paths. If you have a mixed platform environment, you can use a variable ${CONF_OR_INI} for the Kerberos configuration file. Security configuration will expand it to “ini” for Windows or “conf” for non-Windows platforms For example:

```
${WAS_INSTALL_ROOT}\etc\krb5\krb5.${CFG_OR_INI}
```

Note: The configureSpnego and validateSpnegoConfig commands verify the krb5Config and krb5Keytab files only when SPNEGO is enabled. If SPNEGO is not enabled, these commands only verify that the krb5Config and krb5Keytab files exist. This allows you to configure SPNEGO without enabling.

Unconfigure SPNEGO web authentication

Use the unconfigureSpnego command to unconfigure SPNEGO web authentication in the security configuration.

At the wsadmin prompt, enter the following command for help:

```
wsadmin>$AdminTask help unconfigureSpnego
```

Show SPNEGO web authentication

Use the showSPNEGO command to display the SPNEGO web authentication in the security configuration.

At the wsadmin prompt, enter the following command for help:

```
wsadmin>$AdminTask help showSpnego
```

Validate Kerberos configuration

Use the validateKrbConfig command to validate the Kerberos configuration data either in the global security file `security.xml` or specified as an input parameter.

At the wsadmin prompt, enter the following command for help:

```
wsadmin>$AdminTask help validateKrbConfig
```

You can use the following parameters with the validateKrbConfig command:

Table 64. Command parameters.

This table describes parameters for the validateKrbConfig command.

Option	Description
<checkConfigOnly>	Checks the Kerberos configuration without validating, You must use global security for this check.
<useGlobalSecurityConfig>	Uses the Global Security configuration data, security.xml, instead of input parameters.
<validateKrbRealm>	Validates the Kerberos realm against the default Kerberos realm in the Kerberos configuration file (krb5.ini or krb5.conf).
<serverId>	Specifies the server identity that is used for internal process communications.
<serverIdPassword>	Specifies the password that is used for the server identity.
<krb5Spn>	Specifies the Kerberos service principal name in the Kerberos keytab file.
<krb5Config >	This parameter is required. It supplies the directory location and file name of the configuration (krb5.ini or krb5.conf) file.
<krb5Keytab>	This parameter is optional. It supplies the directory location and file name of the Kerberos keytab file. If you do not specify this parameter, the default keytab in the Kerberos configuration file is used.
<krb5Realm >	This parameter is required. It specifies the value for the Kerberos realm name.

Note: WebSphere variables can be used to specify the krb5Config and krb5Keytab file paths. If you have a mixed platform environment, you can use a variable ${CONF_OR_INI} for the Kerberos configuration file. Security configuration will expand it to `ini` for Windows or `conf` for non-Windows platforms. For example:

```
${WAS_INSTALL_ROOT}\etc\krb5\krb5.${CFG_OR_INI}
```

Note: To validate the Kerberos configuration in the global security configuration file `security.xml`, run validateKrbConfig with no parameters or with useGlobalSecurityConfig set to `true`. To validate the Kerberos configuration with input parameters, set useGlobalSecurityConfig and checkConfigOnly to `false` and specify values for krb5Spn, krb5Config, krb5Keytab, and krb5Realm.

SPNEGO web authentication filter commands

Use **wsadmin** commands to add, modify, delete, or show Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Web authentication filters in the security configuration.

Add SPNEGO web authentication filter

Use the **addSpnegoFilter** command to add a new SPNEGO web authentication filter in the security configuration.

At the **wsadmin** prompt, enter the following command for help:

```
wsadmin>$AdminTask help addSpnegoFilter
```

:

Table 65. Command parameters. You can use the following parameters with the **addSpnegoFilter** *command*

Option	Description
<hostName>	This parameter is required. Use to supply a fully-qualified host name.
<krb5Realm>	This parameter is not required. Use to supply a Kerberos realm name. If the krb5Realm parameter is not specified, the default Kerberos realm name in the Kerberos configuration file is used.
<filterCriteria>	This parameter is not required. Use to supply the HTTP request filter rules. If the filterCriteria parameter is not specified, all of the HTTP requests are authenticated by SPNEGO.
<filterClass>	This parameter is not required. Use to supply the HTTP request filter rules. If the filterClass parameter is not specified, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used.
<trimUserName>	This parameter is not required. Use to indicate whether the Kerberos realm name is to be removed from the Kerberos principal name.
<enabledGssCredDelegate>	This parameter is not required. Use to indicate whether to extract and place the client GSS delegation credential in the subject. The default value is `true`.
<spnegoNotSupportedPage>	This parameter is not required. Use to supply the uniform resource identifier (URI) of the resource with a response to be used when SPNEGO is not supported. If this parameter is not specified, the default SPNEGO not supported error page is used.
<ntlmTokenReceivedPage>	This parameter is not required. Use to supply the URI of the resource with a response to be used when an NT LAN manager (NTLM) token is received. If this parameter is not specified, the default NTLM token received error page is used.

The following is an example of the **addSpnegoFilte**r command:

```
wsadmin>$AdminTask addSpnegoFilter {
  -hostName ks.austin.ibm.com
  -krb5Realm WSSEC.AUSTIN.IBM.COM}
```

Modify SPNEGO web authentication filter

Use the **modifySpnegoFilter** command to modify SPNEGO filter attributes in the security configuration.

At the **wsadmin** prompt, enter the following command for help:

```
wsadmin>$AdminTask help modifySpnegoFilter
```

Table 66. Command parameters. You can use the following parameters with the **modifySpnegoFilter** *command:*

Option	Description
<hostName>	This parameter is required. Use to supply a long host name. The hostname is an identifier, so you can not modify the hostname.

Table 66. Command parameters (continued). You can use the following parameters with the **modifySpnegoFilter** *command:*

Option	Description
<krb5Realm>	This parameter is not required. Use to supply a Kerberos realm name. If the krb5Realm parameter is not specified, the default Kerberos realm name in the Kerberos configuration file is used.
<filterCriteria>	This parameter is not required. Use to supply the HTTP request filter rules. If the filterCriteria parameter is not specified, all of the HTTP requests are authenticated by SPNEGO. **Note:** For more information about filter criteria, read the topic Enabling and configuring SPNEGO web authentication using the administrative console.
<filterClass>	This parameter is not required. Use to supply the HTTP request filter rules. If the filterClass is not specified, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used.
<trimUserName>	This parameter is not required. Use to indicate whether the Kerberos realm name is to be removed from the Kerberos principal name.
<enabledGssCredDelegate>	This parameter is not required. Use to indicate whether to extract and place the client GSS delegation credential in the subject. The default value is `true`.
<spnegoNotSupportedPage>	This parameter is not required. Use to supply the URI of the resource with a response to be used when SPNEGO is not supported. If this parameter is not specified, the default SPNEGO not supported error page is used.
<ntlmTokenReceivedPage>	This parameter is not required. Use to supply the URI of the resource with a response to be used when an NTLM token is received. If this parameter is not specified, the default NTLM token received error page is used.

The following is an example of the **modifySpnegoFilter** command:

```
wsadmin>$AdminTask modifySpnegoFilter {
  -hostName ks.austin.ibm.com
  -krb5Realm WSSEC.AUSTIN.IBM.COM}
```

Delete SPNEGO web authentication filter

Use the **deleteSpnegoFilter** command to remove SPNEGO a web authentication filter from the security configuration. If a host name is not specified, all of the SPNEGO web authentication filters are removed.

At the **wsadmin** prompt, enter the following command for help:

```
wsadmin>$AdminTask help deleteSpnegoFilter
```

Table 67. Command parameters. You can use the following parameter with the **deleteSpnegoFilter** *command:*

Option	Description
<hostname>	This parameter is required. If the hostname is not specified, all of the SPNEGO web authentication filters are deleted.

The following is an example of the **deleteSpnegoFilter** command:

```
wsadmin> $AdminTask deleteSpnegoFilter {-hostName ks.austin.ibm.com}
```

Show SPNEGO web authentication filter

Use the **showSpnegoFilter** command to display a SPNEGO web authentication filter in the security configuration. If a host name is not specified, all of the SPNEGO filters are displayed.

At the **wsadmin** prompt, enter the following command for help:

```
wsadmin>$AdminTask help showSpnegoFilter
```

Table 68. Command parameters. You can use the following parameter with the **showSpnegoFilter** *command:*

Option	Description
<hostname>	This parameter is optional. If a long host name is not specified, all of the SPNEGO web authentication filters are displayed.

The following is an example of the **showSpnegoFilter** command:

```
wsadmin> $AdminTask showSpnegoFilter {-hostName ks.austin.ibm.com}
```

Kerberos authentication commands

Use **wsadmin** commands to create, modify or delete Kerberos as the authentication mechanism for WebSphere Application Server.

Create Kerberos authentication mechanism

Note:

The following items are required before you attempt to use the **createKrbAuthMechanism** command to create the KRB5 authentication mechanism security object field in the security configuration file:

- If you do not already have a Kerberos configuration file (**krb5.ini** or **krb5.conf**), use the **createkrbConfigFile** command task to create the Kerberos configuration file. Read about creating a Kerberos configuration file for more information.
- You must have a Kerberos keytab file **(krb5.keytab**) that contains a Kerberos service principal name (SPN), `<service name>/<fully qualified hostname>@KerberosRealm`, for each machine that run WebSphere application servers. The service name can be anything; the default value is `WAS`.

  For example, if you have two application server machines, `host1.austin.ibm.com` and `host2.austin.ibm.com`, the Kerberos keytab file must contain the `<service name>/host1.austin.ibm.com` and `service name>/host2.austin.ibm.com` SPNs and their Kerberos keys.

Use the **createKrbAuthMechanism** command to create the KRB5 authentication mechanism security object field in the security configuration file.

At the **wsadmin** prompt, enter the following command:

```
$AdminTask help createKrbAuthMechanism
```

Table 69. Command parameters. You can use the following parameters with the **createKrbAuthMechanism** *command.*

Option	Description
<krb5Realm>	This parameter is optional. It indicates the Kerberos realm name. If you do not specify this parameter, the default Kerberos realm in the Kerberos configuration file is used.
<krb5Config>	This parameter is required. It indicates the directory location and file name of the configuration (krb5.ini or krb5.conf) file.
<krb5Keytab>	This parameter is optional. It indicates the directory location and file name of the Kerberos keytab file. If you do not specify this parameter, the default keytab in the Kerberos configuration file is used.
<serviceName>	This parameter is required. It indicates the Kerberos service name. The default Kerberos service name is `WAS`.
<trimUserName>	This parameter is optional. It removes the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. This parameter is optional. The default value is `true`.
<enabledGssCredDelegate>	This parameter is not required. Use to indicate whether to extract and place the client GSS delegation credential in the subject. The default value is `true`.
<allowKrbAuthForCsiInbound>	This parameter is optional. It enables Kerberos authentication mechanism for Common Secure Interoperability (CSI) inbound. The default value is `true`.
<allowKrbAuthForCsiOutbound>	This parameter is required. It enables Kerberos authentication mechanism for CSI outbound. The default value is `true`.

Note: The Kerberos configuration file name and Kerberos keytab filename path do not have to be absolute paths. You can use WebSphere variables for the paths instead. If you have a mixed platform environment, you can use a variable ${CONF_OR_INI} for the Kerberos configuration file. Security configuration will expand it to "ini" for Windows or "conf" for non-Windows platforms For example:

```
${WAS_INSTALL_ROOT}\etc\krb5\krb5.${CFG_OR_INI}
```

The following is an example of the createKrbAuthMechanism command:

```
wsadmin>$AdminTask createKrbAuthMechanism {
  -krb5Realm  WSSEC.AUSTIN.IBM.COM
  -krb5Config C:\\WINNT\\krb5.ini
  -krb5Keytab C:\\WINNT\\krb5.keytab
  -serviceName WAS }
```

Modify Kerberos authentication mechanism

Use the **modifyKrbAuthMechanism** command to make changes to the KRB5 authentication mechanism security object field in the security configuration file.

At the **wsadmin** prompt, enter the following command:

```
$AdminTask help modifyKrbAuthMechanism
```

Table 70. Command parameters. You can use the following parameters with the **modifyKrbAuthMechanism** *command.*

Option	Description
<krb5Realm>	This parameter is optional. It indicates the Kerberos realm name. If you do not specify this parameter, the default Kerberos realm in the Kerberos configuration file is used.
<krb5Config>	This parameter is required. It indicates the directory location and file name of the configuration (krb5.ini or krb5.conf) file.
<krb5Keytab>	This parameter is optional. It indicates the directory location and file name of the Kerberos keytab file. If you do not specify this parameter, the default keytab in the Kerberos configuration file is used.
<serviceName>	This parameter is required. It indicates the Kerberos service name. The default Kerberos service name is `WAS`.
<trimUserName>	This parameter is optional. It removes the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. This parameter is optional. The default value is `true`.
<enabledGssCredDelegate>	This parameter is not required. Use to indicate whether to extract and place the client Kerberos and GSS delegation credential in the Kerberos authentication token (KRBAuthnToken). The default value is `true`. **Note:** If this parameter is `true`, and the runtime cannot extract the Kerberos GSS delegation credential, the runtime logs a warning message.
<allowKrbAuthForCsiInbound>	This parameter is optional. It enables Kerberos authentication mechanism for Common Secure Interoperability (CSI) inbound. The default value is `true`.
<allowKrbAuthForCsiOutbound>	This parameter is optional. It enables Kerberos authentication mechanism for CSI outbound. The default value is `true`.

Note: The Kerberos configuration file name and Kerberos keytab filename path do not have to be absolute paths. You can use WebSphere variables for the paths instead. If you have a mixed platform environment, you can use a variable ${CONF_OR_INI} for the Kerberos configuration file. Security configuration will expand it to "ini" for Windows or "conf" for non-Windows platforms For example:

```
${WAS_INSTALL_ROOT}\etc\krb5\krb5.${CFG_OR_INI}
```

The following is an example of the **modifyKrbAuthMechanism** command:

```
wsadmin>$AdminTask modifyKrbAuthMechanism {
   -krb5Realm  WSSEC.AUSTIN.IBM.COM
   -krb5Config C:\\WINNT\\krb5.ini
   -krb5Keytab C:\\WINNT\\krb5.keytab
   -serviceName WAS }
```

Delete Kerberos authentication mechanism

Use the **deleteKrbAuthMechanism** command to remove the KRB5 authentication mechanism security object field in the security configuration file.

At the **wsadmin** prompt, enter the following command:

```
$AdminTask help deleteKrbAuthMechanism
```

The following is an example of the **deleteKrbAuthMechanism** command:

```
wsadmin>$AdminTask deleteKrbAuthMechanism
```

Set active authentication mechanism

Use the **setActiveAuthMechanism** command to set the active authentication mechanism attribute in the security configuration.

At the **wsadmin** prompt, enter the following command:

```
$AdminTask help setActiveAuthMechanism
```

Table 71. Command parameters. You can use the following parameter with the **setActiveAuthMechanism** *command.*

Option	Description
<authMechanismType>	This parameter is not required. It indicates the authentication mechanism type. The default is `KRB5`.

The following is an example of the **setActiveAuthMechanism** command:

```
wsadmin> $AdminTask setActiveAuthMechanism {-authMechanismType KRB5 }
```

Chapter 7. Scripting for Service integration

This page provides a starting point for finding information about service integration.

Service integration provides asynchronous messaging services. In asynchronous messaging, producing applications do not send messages directly to consuming applications. Instead, they send messages to destinations. Consuming applications receive messages from these destinations. A producing application can send a message and then continue processing without waiting until a consuming application receives the message. If necessary, the destination stores the message until the consuming application is ready to receive it.

Printing a summary of the runtime state of all messaging engines running in a cell

You can use scripting to list the details of all messaging engines in all buses.

About this task

The following script can be used to print a summary in XML form of the runtime state of all messaging engines running in a cell. It goes down as far as the depths of individual items, such as queue points, remote queue points, publication points, subscriptions, mediation points, and link transmitters. However, this script does not print out details of individual messages. It should be run against the deployment manager to list out all messaging engines in all buses.

Procedure

1. Create your script using a text editor.
2. Run the script against the deployment manager of the cell:

   ```
   DMGR_PROFILE/bin/wsadmin.sh -lang jython -f printSIBusSummary.py output.xml
   ```

Example

This example provides a sample script, with sample output.

```
# Sample program
#  * (C) COPYRIGHT International Business Machines Corp., 2008, 2009
#  * All Rights Reserved * Licensed Materials - Property of IBM
#  *
#  * This sample program is provided AS IS and may be used, executed,
#  * copied and modified without royalty payment by customer
#  *
#  * (a) for its own instruction and study,
#  * (b) in order to develop applications designed to run with an IBM
#  *     WebSphere product for the customer's own internal use.
#
# Version: 1.01
#
# Information:
#  This script prints an XML summary of the runtime information available
#  for all queue points, publication points, mediation points, SIBus
#  links and WMQ links.
#  The script should be run against the deployment manager of the cell,
#  so that output is available from all running messaging engines in all buses.
#  The script collects the same information that can be displayed in via the
#  administrative console.
#  The following output is only available when the script is executed against
#  a WebSphere Application Server
Version 7.0 or later environment:
#  - Service integration bus link information
#  - WebSphere MQ link information
#
# Usage:
#  UNIX, Linux, z/OS:
#    <DMGR_PROFILE>/bin/wsadmin.sh -lang jython -f printSIBusSummary.py <OUTFILE.XML>
#  Windows:
#    <DMGR_PROFILE>\bin\wsadmin.bat -lang jython -f printSIBusSummary.py <OUTFILE.XML>

# Class to print an error out in full (with stack) to STDERR, as well as a short summary
# within the XML output. An integer is assigned to each error, to help match up the two.
```

```
class ErrorTracker:
  errorCount = 0
  # Helper method to print exception details as an error attribute in a tag
  def printErrorAndCloseTag(self, exception_tuple, oneLine):
    self.errorCount += 1
    # Print the error to STDERR
    sys.stderr.write("ERROR [" + str(self.errorCount) + "]:\n")
    sys.excepthook(exception_tuple[0],exception_tuple[1],exception_tuple[2])
    # Print a summary of the error to the output file
    out.write(' error="[' + str(self.errorCount) + ']: ')
    out.write(str(exception_tuple[0]).strip())
    if exception_tuple[1] != None: out.write(': ' + str(exception_tuple[1]).strip())
    if oneLine == 1: out.write('"/>\n')
    else: out.write('">\n')
# Single global instance of error tracker
et = ErrorTracker()

# Helper method to get a JMX attribute in string form
def getStrAttr(mBean, attr):
  val = AdminControl.getAttribute_jmx(mBean, attr)
  if val == None: val = ''
  else: val = str(val)
  return val

# Helper method to get the return value of JMX method in string form
def getStrReturnVal(mBean, method):
  val = AdminControl.invoke_jmx(mBean, method, [], [])
  if val == None: val = ''
  else: val = str(val)
  return val

# Helper toString method to handle None values as empty strings
def toStr(val):
  if val == None: val = ''
  else: val = str(val)
  return val

# Messaging engine class wraps an ME, maps its name to/from a UUID,
# and contains all the methods we used to print out the runtime state
# of that messaging engines
class MessagingEngine:
  "A class for printing a runtime summary of a messaging engine"
  # Constructor
  def __init__(self, bus, name, uuid):
    self.bus = bus
    self.name = name
    self.uuid = uuid

  # Method to print a summary of the runtime state of this messaging engine
  # - this is a the entry point into the class (other methods are logically private)
  def printRuntimeStateXML(self, indent):
    # Print the start of the tag (leaving room for additional properties)
    out.write(indent + '<MessagingEngine name="' + self.name + '" uuid="' + self.uuid + '"')
    meMBean = None
    try:
      # First lookup our MBean
      meLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMessagingEngine,name=' + self.name + ',*')
      meMBeans = AdminControl.queryNames_jmx(meLookupName, None)
      if (meMBeans == None) or (meMBeans.size() == 0):
        # Just an empty messaging engine
        out.write(' state="Unknown (no MBean found)">\n')
      elif meMBeans.size() == 1:
        # Save a ref to the MBean
        meMBean = meMBeans[0]
        # Complete the entry tag
        out.write(' state="' + AdminControl.invoke_jmx(meMBean, "state", [], []) + '"')
        out.write(' activeServer="' + meMBean.getKeyProperty("process") + '"')
        out.write('>\n')
      else:
        # We only expect to zero/one MBean
        raise Exception, "Found " + str(len(meMBeans)) + " MBeans for messaging engine. Expected 1"
    except:
       et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Iterate through the contents
    if (meMBean != None):
      self.printQueuePointsXML(meMBean, indent + " ")
      self.printRemoteQueuePointsXML(meMBean, indent + " ")
      self.printMediationPointsXML(meMBean, indent + " ")
      self.printRemoteMediationPointsXML(meMBean, indent + " ")
      self.printPublicationPointsXML(meMBean, indent + " ")
      self.printBusLinks(meMBean, indent + " ")
      self.printWMQLinks(meMBean, indent + " ")
    # Complete our tag
    out.write(indent + "</MessagingEngine>\n")

  # Print a summary of all the queue points for the messaging engine,
  # using the supplied ME MBean looked up by the caller
  def printQueuePointsXML(self, meMBean, indent):
    qpMBeans = None
    out.write(indent + "<QueuePoints")
```

```
    try:
      # Get a list of queue point MBeans
      qpLookupName = AdminControl.makeObjectName('WebSphere:type=SIBQueuePoint,SIBMessagingEngine='
+ self.name + ',*')      qpMBeans = AdminControl.queryNames_jmx(qpLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each one
    for qpMBean in qpMBeans:
      rqps = {} # Directory of all RQPs for this queue point
      oneLineTag = 0
      out.write(indent + " <QueuePoint")
      try:
        out.write(' name="'  +  qpMBean.getKeyProperty("name") + '@' + self.name + '"')
        qpState = getStrAttr(qpMBean, "state")
        out.write(' state="' +  qpState + '"')
        out.write(' depth="' +  getStrAttr(qpMBean, "depth") + '"')
        hmt = AdminControl.getAttribute_jmx(qpMBean, "highMessageThreshold")
        if (hmt != None) and (hmt == java.lang.Long.MAX_VALUE): hmt = "MAX_VALUE"
        out.write(' highMessageThreshold="' +  str(hmt) + '"')
        out.write(' sendAllowed="' +  getStrAttr(qpMBean, "sendAllowed") + '"')
        # Only attempt to get additional details for active queue points
        if qpState == 'ACTIVE':
          # Get a list of inbound receivers, for remote queue points
          inboundReceivers = AdminControl.invoke_jmx(qpMBean, "listInboundReceivers", [], [])
          # Get a list of remote consumer transmitters, for remote queue points
          consumerTransmitters = AdminControl.invoke_jmx(qpMBean, "listRemoteConsumerTransmitters", [], [])
          # Add RQPs for all inbound receivers
          for ir in inboundReceivers:
            uuid = ir.getRemoteEngineUuid()
            if rqps.has_key(uuid): rqp = rqps[uuid]
            else: rqp = KnownRemoteQueuePoint(qpMBean, uuid)
            rqps[uuid] = rqp
            rqp.inboundReceiver = ir
          # Add RQPs for all consumer transmitters
          for ct in consumerTransmitters:
            uuid = ct.getRemoteEngineUuid()
            if rqps.has_key(uuid): rqp = rqps[uuid]
            else: rqp = KnownRemoteQueuePoint(qpMBean, uuid)
            rqps[uuid] = rqp
            rqp.consumerTransmitter = ct
        # If we do not have any RQPs then we can terminate the tag on this line
        if len(rqps.keys()) == 0:
          out.write('/>\n')
          oneLineTag = 1
        else:
          out.write('>\n') # We need a full tag
      except:
        et.printErrorAndCloseTag(sys.exc_info(), 0)
      # Process each RQP we found
      for rqpUuid in rqps.keys():
        rqp = rqps[rqpUuid]
        rqp.printSummaryXML(indent + "  ")
      # Complete our QueuePoint tag
      if oneLineTag == 0: out.write(indent + " </QueuePoint>\n")
    # Complete our QueuePoints tag
    out.write(indent + "</QueuePoints>\n")

  # Print a summary of all the remote queue points for the messaging engine,
  # using the supplied ME MBean looked up by the caller
  def printRemoteQueuePointsXML(self, meMBean, indent):
    rqpMBeans = None
    out.write(indent + "<RemoteQueuePoints")
    try:
      # Get a list of remote queue point MBeans
      rqpLookupName = AdminControl.makeObjectName('WebSphere:type=SIBRemoteQueuePoint,
SIBMessagingEngine=' + self.name + ',*')      rqpMBeans = AdminControl.queryNames_jmx(rqpLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each one
    for rqpMBean in rqpMBeans:
      oneLineTag = 0
      out.write(indent + " <RemoteQueuePoint")
      try:
        out.write(' name="'  +  rqpMBean.getKeyProperty("name") + '@' + self.name + '"')
        remoteMEuuid = getStrAttr(rqpMBean, "remoteMessagingEngineUuid")
        if (mesByUUID.has_key(remoteMEuuid)): remoteMEName = mesByUUID[remoteMEuuid].name
        else: remoteMEName = "Unknown"
        out.write(' remoteME="' +  remoteMEName + '"')
        out.write(' remoteMEUUID="' +  remoteMEuuid + '"')

        # Get outbound transmitter details, if one exists
        currentOutboundMessages = 0
        outboundMessagesSent = 0
        outboundTransmitter = AdminControl.invoke_jmx(rqpMBean, "getOutboundTransmitter", [], [])
        if (outboundTransmitter != None):
          currentOutboundMessages = outboundTransmitter.getDepth()
          outboundMessagesSent = outboundTransmitter.getNumberOfMessagesSent()
        out.write(' currentOutboundMessages="' + str(currentOutboundMessages) + '"')
```

```
        out.write(' outboundMessageSent="' + str(outboundMessagesSent) + '"')

        # Get remote consumer receiver, if one exists
        remoteConsumerReceiver = AdminControl.invoke_jmx(rqpMBean, "getRemoteConsumerReceiver", [], [])
        currentMessageRequests = 0
        completedMessageRequests = 0
        messageRequestsIssued = 0
        if (remoteConsumerReceiver != None):
          currentMessageRequests = remoteConsumerReceiver.getNumberOfActiveRequests()
          completedMessageRequests = remoteConsumerReceiver.getNumberOfCompletedRequests()
          messageRequestsIssued = remoteConsumerReceiver.getNumberOfRequestsIssued()
        out.write(' currentMessageRequests="' + str(currentMessageRequests) + '"')
        out.write(' completedMessageRequests="' + str(completedMessageRequests) + '"')
        out.write(' messageRequestsIssued="' + str(messageRequestsIssued) + '"')

        # Always one line for remote queue points
        out.write('/>\n')
      except:
        et.printErrorAndCloseTag(sys.exc_info(), 1)
    # Complete our QueuePoints tag
    out.write(indent + "</RemoteQueuePoints>\n")

  # Print a summary of all the mediation points for the messaging engine,
  # using the supplied ME MBean looked up by the caller
  def printMediationPointsXML(self, meMBean, indent):
    mpMBeans = None
    out.write(indent + "<MediationPoints")
    try:
      # Get a list of mediation point MBeans
      mpLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMediationPoint,SIBMessagingEngine='
+ self.name + ',*')      mpMBeans = AdminControl.queryNames_jmx(mpLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each one
    for mpMBean in mpMBeans:
      out.write(indent + " <MediationPoint")
      try:
        out.write(' name="'  +  mpMBean.getKeyProperty("name") + '@' + self.name + '"')
        mpState = getStrAttr(mpMBean, "currentState")
        out.write(' status="' +  mpState + '"')
        out.write(' depth="' +  getStrAttr(mpMBean, "depth") + '"')
        hmt = AdminControl.getAttribute_jmx(mpMBean, "highMessageThreshold")
        if (hmt != None) and (hmt == java.lang.Long.MAX_VALUE): hmt = "MAX_VALUE"
        out.write(' highMessageThreshold="' +  str(hmt) + '"')
        out.write(' sendAllowed="' +  getStrAttr(mpMBean, "sendAllowed") + '"')
        out.write('/>\n')
      except:
        et.printErrorAndCloseTag(sys.exc_info(), 1)
    # Complete our QueuePoints tag
    out.write(indent + "</MediationPoints>\n")

  # Print a summary of all the remote mediation points for the messaging engine,
  # using the supplied ME MBean looked up by the caller
  def printRemoteMediationPointsXML(self, meMBean, indent):
    rmpMBeans = None
    out.write(indent + "<RemoteMediationPoints")
    try:
      # Get a list of remote mediation point MBeans
      rmpLookupName = AdminControl.makeObjectName('WebSphere:type=SIBRemoteMediationPoint,SIBMessagingEngine=' +
self.name + ',*')      rmpMBeans = AdminControl.queryNames_jmx(rmpLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each one
    for rmpMBean in rmpMBeans:
      oneLineTag = 0
      out.write(indent + " <RemoteMediationPoint")
      try:
        out.write(' name="'  +  rmpMBean.getKeyProperty("name") + '@' + self.name + '"')
        remoteMEuuid = getStrAttr(rmpMBean, "remoteMessagingEngineUuid")
        if (mesByUUID.has_key(remoteMEuuid)): remoteMEName = mesByUUID[remoteMEuuid].name
        else: remoteMEName = "Unknown"
        out.write(' remoteME="' +  remoteMEName + '"')
        out.write(' remoteMEUUID="' +  remoteMEuuid + '"')

        # Get outbound transmitter details, if one exists
        currentOutboundMessages = 0
        outboundMessagesSent = 0
        outboundTransmitter = AdminControl.invoke_jmx(rmpMBean, "getOutboundTransmitter", [], [])
        if (outboundTransmitter != None):
          currentOutboundMessages = outboundTransmitter.getDepth()
          outboundMessagesSent = outboundTransmitter.getNumberOfMessagesSent()
        out.write(' currentOutboundMessages="' + str(currentOutboundMessages) + '"')
        out.write(' outboundMessageSent="' + str(outboundMessagesSent) + '"')

        # Always one line for remote mediation points
        out.write('/>\n')
      except:
        et.printErrorAndCloseTag(sys.exc_info(), 1)
```

```
    # Complete our MediationPoints tag
    out.write(indent + "</RemoteMediationPoints>\n")

  # Print a summary of all the publication points for the messaging engine,
  # using the supplied ME MBean looked up by the caller
  def printPublicationPointsXML(self, meMBean, indent):
    ppMBeans = None
    out.write(indent + "<PublicationPoints")
    try:
      # Get a list of publication point MBeans
      ppLookupName = AdminControl.makeObjectName('WebSphere:type=SIBPublicationPoint,SIBMessagingEngine='
 + self.name + ',*')      ppMBeans = AdminControl.queryNames_jmx(ppLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each one
    for ppMBean in ppMBeans:
      inboundReceivers = []
      subscriptions = []
      out.write(indent + " <PublicationPoint")
      depth = None
      try:
        depth = getStrAttr(ppMBean, "depth")
      except:
        # Attribute does not exist
        pass
      try:
        out.write(' name="'  +  ppMBean.getKeyProperty("name") + '@' + self.name + '"')
        if depth != None: out.write(' depth="' +  depth + '"')
        hmt = AdminControl.getAttribute_jmx(ppMBean, "highMessageThreshold")
        if (hmt != None) and (hmt == java.lang.Long.MAX_VALUE): hmt = "MAX_VALUE"
        out.write(' highMessageThreshold="' +  str(hmt) + '"')
        out.write(' sendAllowed="' +  getStrAttr(ppMBean, "sendAllowed") + '"')
        # Get a list of inbound receivers (remote publication points)
        inboundReceivers = AdminControl.invoke_jmx(ppMBean, "listInboundReceivers", [], [])
        # Get a list of subscriptions
        subscriptions = AdminControl.invoke_jmx(ppMBean, "getSubscriptions", [], [])
        # Complete the tag
        out.write('>\n')
      except:
        et.printErrorAndCloseTag(sys.exc_info(), 0)
      # Run through each remote publication point (inbound receiver)
      for ir in inboundReceivers:
        out.write(indent + '  <RemotePublicationPoint')
        try:
          remoteMEuuid = ir.getRemoteEngineUuid()
          if mesByUUID.has_key(remoteMEuuid): remoteMEname = mesByUUID[remoteMEuuid].name
          else: remoteMEname = "Unknown"
          out.write(' me="' + remoteMEname + '"')
          out.write(' meUUID="' + remoteMEuuid + '"')
          out.write(' currentInboundMessages="' + str(ir.getDepth()) + '"')
          out.write(' inboundMessagesReceived="' + str(ir.getNumberOfMessagesReceived()) + '"')
          out.write('/>\n')
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 1)
      # Run through each subscription
      for sub in subscriptions:
        oneLineTag = 0
        rsps = [] # remote subscription points
        out.write(indent + '  <Subscription')
        try:
          out.write(' subscriberId="' + toStr(sub.getSubscriberId()) + '"')
          out.write(' depth="' + str(sub.getDepth()) + '"')
          # Write any selector
          selector = sub.getSelector()
          if selector != None: out.write(' selector="' + sub.getSelector() + '"')
          # Write topics
          topics = sub.getTopics()
          if (topics != None):
            out.write(' topics="')
            sep = ''
            for topic in topics:
              if (topic == None): topic = ''
              out.write(topic + sep)
              sep = ','
            out.write('"')
          # Get a list of remote subscription points
          rsps = AdminControl.invoke_jmx(ppMBean, "listRemoteConsumerTransmitters", [sub],
['com.ibm.websphere.sib.admin.SIBSubscription'])
          # Check if we have children, or can just close the tag here
          if len(rsps) == 0:
            out.write('/>\n')
            oneLineTag = 1
          else: out.write('>\n')
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 0)
        # Do we have remote sub points?
        for rsp in rsps:
          out.write(indent + '   <KnownRemoteSubscriptionPoint')
          try:
```

```
            remoteMEuuid = rsp.getRemoteEngineUuid()
            if mesByUUID.has_key(remoteMEuuid): remoteMEname = mesByUUID[remoteMEuuid].name
            else: remoteMEname = "Unknown"
            out.write(' me="' + remoteMEname + '"')
            out.write(' meUUID="' + remoteMEuuid + '"')
            out.write(' currentMessageRequests="' + str(rsp.getDepth()) + '"')
            out.write(' completedMessageRequests="' + str(rsp.getNumberOfCompletedRequests()) + '"')
            out.write(' messageRequestsReceived="' + str(rsp.getNumberOfRequestsReceived()) + '"')
            out.write('/>\n')
          except:
            et.printErrorAndCloseTag(sys.exc_info(), 1)
        # Close the subscription tag if required
        if oneLineTag == 0: out.write(indent + '  </Subscription>\n')
      # Complete our PublicationPoint tag
      out.write(indent + " </PublicationPoint>\n")
    # Complete our PublicationPoints tag
    out.write(indent + "</PublicationPoints>\n")

  # Print a summary of all the SIBus links hosted on this messaging engine.
  # using the supplied ME MBean looked up by the caller
  def printBusLinks(self, meMBean, indent):
    glMBeans = None
    out.write(indent + "<BusLinks")
    try:
      # Get a list of link transmitter MBeans - will be empty for <V7.0 MEs
      glLookupName = AdminControl.makeObjectName('WebSphere:type=SIBGatewayLink,SIBMessagingEngine=' +
self.name + ',*')      glMBeans = AdminControl.queryNames_jmx(glLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Keep track of all link target UUIDs we've seen as local links, to exclude from the remote transmitter list
    localLinkUuids = {}
    # Run through each transmitter
    for glMBean in glMBeans:
      oneLineTag = 0
      linkReceivers = []
      targetUuid = glMBean.getKeyProperty("targetUuid")
      localLinkUuids[targetUuid] = 1
      out.write(indent + " <BusLink")
      # First check we can query the foreign bus name... if this fails we are
      # talking to a < V7 messaging engine
      foreignBusName = None
      oldVersion = 0
      try:
        foreignBusName = getStrReturnVal(glMBean, "getForeignBusName")
      except:
        oldVersion = 1
      # Get the name (sometimes the returned name includes quotes)
      virtualLinkName = toStr(glMBean.getKeyProperty("name"))
      if not (virtualLinkName.find('"') == 0): virtualLinkName = '"' + virtualLinkName + '"'
      # Print the correct information based on the version
      if oldVersion:
        try:
          out.write(' name=' + virtualLinkName)
          out.write(' state="' +  getStrReturnVal(glMBean, "getStatus") + '"')
          out.write('>\n')
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 0)
      else:
        try:
          out.write(' name=' + virtualLinkName )
          out.write(' foreignBus="' + foreignBusName + '"')
          stateString = AdminControl.invoke_jmx(glMBean, "getStatus", [], [])
          if stateString != None:
            out.write(' state="' +  stateString + '"')
          linkReceivers = AdminControl.invoke_jmx(glMBean, "listLinkReceivers", [], [])
          if linkReceivers == None: linkReceivers = []
          out.write('>\n')
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 0)
        # Print out the link receivers for this link
        for lr in linkReceivers:
          out.write(indent + "  <LinkReceiver")
          try:
            out.write(' state="' +  lr.getState() + '"')
            receiverType = lr.getReceiverType()
            remoteMEuuid = lr.getForeignEngineUuid()
            if mesByUUID.has_key(remoteMEuuid): remoteMEname = mesByUUID[remoteMEuuid].name
            else: remoteMEname = "Unknown"
            out.write(' me="' +  remoteMEname + '"')
            out.write(' meUUID="' +  remoteMEuuid + '"')
            out.write(' receiverType="' +  receiverType + '"')
            if receiverType == "PUBLICATION":
              out.write(' topicSpace="' +  toStr(lr.getTargetDestination()) + '"')
            out.write(' currentInboundMessages="' +  str(lr.getDepth()) + '"')
            out.write(' messagesReceived="' +  str(lr.getNumberOfMessagesReceived()) + '"')
            timeSinceLastMessageReceived = lr.getTimeSinceLastMessageReceived()
            if timeSinceLastMessageReceived > 0:
              out.write(' timeSinceLastMessageReceived="' +  str(timeSinceLastMessageReceived) + 'ms"')
            out.write(indent + "/>\n")
```

```
          except:
            et.printErrorAndCloseTag(sys.exc_info(), 1)
        # Print out the link transmitters for this link
        self.printLinkTransmittersXML(meMBean, targetUuid, {}, indent + '  ')
      # End the link tag
      out.write(indent + ' </BusLink>\n')
    # Just in case we have any orphaned link mBeans, print these out here
    self.printLinkTransmittersXML(meMBean, None, localLinkUuids, indent + ' ')
    # Complete our SIBLinks tag
    out.write(indent + "</BusLinks>\n")

  # Print a summary of SIBus link transmitters for the messaging engine,
  # using the supplied ME MBean looked up by the caller.
  # Either prints all transmitters with a particular target UUID, or
  # prints all transmitters excluding keys that exist in the excludeUuids hash.
  def printLinkTransmittersXML(self, meMBean, targetUuid, excludeUuids, indent):
    ltMBeans = None
    try:
      # Get a list of link transmitter MBeans - will be empty for <V7.0 MEs
      lookupString = 'WebSphere:type=SIBLinkTransmitter,SIBMessagingEngine=' + self.name
      if targetUuid != None: lookupString += ',targetUuid=' + targetUuid
      lookupString += ',*'
      ltLookupName = AdminControl.makeObjectName(lookupString)
      ltMBeans = AdminControl.queryNames_jmx(ltLookupName, None)
    except:
      out.write('<SIBLinkTransmitters')
      et.printErrorAndCloseTag(sys.exc_info(), 1)
    # Run through each transmitter
    for ltMBean in ltMBeans:
      # Check this one shouldn't be excluded
      transmitterTargetUuid = ltMBean.getKeyProperty("targetUuid")
      if not excludeUuids.has_key(transmitterTargetUuid):
        out.write(indent + "<LinkTransmitter")
        try:
          if (targetUuid == None): out.write(' foreignBus="' + getStrReturnVal(ltMBean, "getForeignBusName") + '"')
          out.write(' state="' +  getStrReturnVal(ltMBean, "getState") + '"')
          out.write(' linkType="' + getStrReturnVal(ltMBean, "getLinkType") + '"')
          transmitterType = getStrReturnVal(ltMBean, "getTransmitterType")
          out.write(' transmitterType="' +  transmitterType + '"')
          if transmitterType == "PUBLICATION":
            out.write(' topicSpace="' +  getStrReturnVal(ltMBean, "getTargetDestination") + '"')
          putInhibited = AdminControl.invoke_jmx(ltMBean, "isPutInhibited", [], [])
          if putInhibited == 0: sendAllowed = 1
          else: sendAllowed = 0
          out.write(' sendAllowed="' + str(sendAllowed) + '"')
          out.write(' currentOutboundMessages="' +  getStrReturnVal(ltMBean, "getDepth") + '"')
          out.write(' messagesSent="' +  getStrReturnVal(ltMBean, "getNumberOfMessagesSent") + '"')
          timeSinceLastMessageSent = AdminControl.invoke_jmx(ltMBean, "getTimeSinceLastMessageSent", [], [])
          if timeSinceLastMessageSent > 0:
            out.write(' timeSinceLastMessageSent="' +  str(timeSinceLastMessageSent) + 'ms"')

          # Always one line for link transmitters
          out.write('/>\n')
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 1)

  # Print a summary of all the WMQ links hosted on this messaging engine
  # using the supplied ME MBean looked up by the caller
  def printWMQLinks(self, meMBean, indent):
    mqlMBeans = None
    out.write(indent + "<WMQLinks")
    try:
      # Get a list of WMQ Link MBeans - will be empty for <V7.0 MEs
      mqlLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMQLink,SIBMessagingEngine=' + self.name + ',*')
      mqlMBeans = AdminControl.queryNames_jmx(mqlLookupName, None)
      out.write('>\n') # Complete the tag as lookup was successful
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 0)
    # Run through each WMQ Link MBean found
    for mqlMBean in mqlMBeans:
      schlMBeans = []
      rchlMBeans = []
      # We may not be able to query msgs received as introduced at v7.0
      msgsReceived = None
      try:
        msgsReceived = getStrReturnVal(mqlMBean, "getTotalLinkMessagesReceived")
      except:
        pass
      out.write(indent + " <WMQLink")
      try:
        # Get the targetUuid
        linkName = mqlMBean.getKeyProperty("name")
        out.write(' name="' +  linkName + '"')
        out.write(' state="' +  getStrReturnVal(mqlMBean, "getOverallStatus") + '"')
        if msgsReceived != None: out.write(' messagesReceived="' +  msgsReceived + '"')
        # Get a list of WMQ link sender channel instances
        schlLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMQLinkSenderChannel,SIBMessagingEngine='
+ self.name + ',name=' + linkName + 'SNDR,*')
        schlMBeans = AdminControl.queryNames_jmx(schlLookupName, None)
        # Get a list of WMQ link receiver channel instances
```

```
        rchlLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMQLinkReceiverChannel,SIBMessagingEngine='
+ self.name + ',name=' + linkName + 'RCVR,*')
        rchlMBeans = AdminControl.queryNames_jmx(rchlLookupName, None)
        out.write('>\n') # Complete the opening tag
      except:
         et.printErrorAndCloseTag(sys.exc_info(), 0)
      # Iterate through the sender channels
      for schlMBean in schlMBeans:
        sxmitMBeans = []
        out.write(indent + "  <SenderChannel")
        try:
          stateObject = AdminControl.invoke_jmx(schlMBean, "getCurrentStatus", [], [])
          channelName = stateObject.getChannelName()
          out.write(' channelName="' + channelName + '"')
          out.write(' state="' + toStr(stateObject.getState()) + '"')
          out.write(' virtualQmgr="' + stateObject.getQueueManager() + '"')
          out.write(' ipAddress="' + toStr(stateObject.getIpAddress()) + '"')
          out.write(' messagesSent="' + toStr(stateObject.getNumberOfMessagesSent()) + '"')
          out.write(' currentLUWID="' + toStr(stateObject.getCurrentLUWID()) + '"')
          out.write(' currentSequenceNo="' + toStr(stateObject.getCurrentSequenceNumber()) + '"')
          out.write(' inDoubt="' + toStr(stateObject.getInDoubt()) + '"')
          # Get a list of WMQ link sender channel transmitter instances
          sxmitLookupName = AdminControl.makeObjectName('WebSphere:type=SIBMQLinkSenderChannelTransmitter,
SIBMessagingEngine=' + self.name + ',name=' + channelName + ',*')
          sxmitMBeans = AdminControl.queryNames_jmx(sxmitLookupName, None)
          out.write('>\n') # Complete the opening tag
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 0)
        # Iterate through the sender channel transmitters
        for sxmitMBean in sxmitMBeans:
          out.write(indent + "   <SenderChannelTransmitter")
          # List of the known link transmitters
          knownLinkTransmitters = []
          oneLineTag = 0
          try:
            out.write(' status="' +  getStrReturnVal(sxmitMBean, "getState") + '"')
            out.write(' currentOutboundMessages="' +  getStrReturnVal(sxmitMBean, "getDepth") + '"')
            out.write(' messagesSent="' +  getStrReturnVal(sxmitMBean, "getNumberOfMessagesSent") + '"')
            timeSinceLastMessageSent = AdminControl.invoke_jmx(sxmitMBean, "getTimeSinceLastMessageSent", [], [])
            if timeSinceLastMessageSent > 0:
              out.write(' timeSinceLastMessageSent="' + str(timeSinceLastMessageSent) + 'ms"')
            # List the known link transmitters
            knownLinkTransmitters = AdminControl.invoke_jmx(sxmitMBean, "listInboundReceivers", [], [])
            # Complete the tag
            if (knownLinkTransmitters == None) or (len(knownLinkTransmitters)) == 0:
              out.write('/>\n') # Complete the one-line tag
              oneLineTag = 1
            else: out.write('>\n')
          except:
            et.printErrorAndCloseTag(sys.exc_info(), 1)
          # Print the known link transmitters
          if (oneLineTag == 0):
            for ir in knownLinkTransmitters:
              out.write(indent + '    <KnownRemoteSenderChannelTransmitter')
              try:
                remoteMEuuid = ir.getRemoteEngineUuid()
                if mesByUUID.has_key(remoteMEuuid): remoteMEname = mesByUUID[remoteMEuuid].name
                else: remoteMEname = "Unknown"
                out.write(' me="' + remoteMEname + '"')
                out.write(' meUUID="' + remoteMEuuid + '"')
                out.write(' currentInboundMessages="' + str(ir.getDepth()) + '"')
                out.write(' inboundMessagesReceived="' + str(ir.getNumberOfMessagesReceived()) + '"')
                out.write('/>\n')
              except:
                et.printErrorAndCloseTag(sys.exc_info(), 1)
            # Close the transmitter tag
            out.write(indent + "   </SenderChannelTransmitter>\n")
        # Close the sender tag
        out.write(indent + "  </SenderChannel>\n")
      # Iterate through the receiver channels
      for rchlMBean in rchlMBeans:
        out.write(indent + "  <ReceiverChannel")
        statusEntries = []
        try:
          out.write(' state="' +  getStrReturnVal(rchlMBean,"getOverallStatus") + '"')
          # List the status of each instance
          statusEntries = AdminControl.invoke_jmx(rchlMBean, "getCurrentStatus", [], [])
          if (statusEntries == None): statusEntries = []
          out.write('>\n') # Complete the opening tag
        except:
          et.printErrorAndCloseTag(sys.exc_info(), 0)
        # Print each entry
        for statusEntry in statusEntries:
          out.write(indent + "   <ReceiverChannelStatus")
          try:
            out.write(' channelName="' +  toStr(statusEntry.getChannelName()) + '"')
            out.write(' state="' +  toStr(statusEntry.getState()) + '"')
            out.write(' qmgr="' +  toStr(statusEntry.getQueueManager()) + '"')
            out.write(' ipAddress="' + toStr(statusEntry.getIpAddress()) + '"')
            out.write(' messagesReceived="' + toStr(statusEntry.getNumberOfMessagesReceived()) + '"')
```

```
            out.write(' currentLUWID="' + toStr(statusEntry.getCurrentLUWID()) + '"')
            out.write(' currentSequenceNo="' + toStr(statusEntry.getCurrentSequenceNumber()) + '"')
            out.write('/>\n') # Complete the tag
          except:
            et.printErrorAndCloseTag(sys.exc_info(), 1)
        # Close the tag
        out.write(indent + "  </ReceiverChannel>\n")
      # Complete our WMQLink tag
      out.write(indent + " </WMQLink>\n")
    # Complete our WMQLinks tag
    out.write(indent + "</WMQLinks>\n")

# A small class to help us agregate consumer-transmitter and
# inbound-receiver information on a queue point, and hence build a
# "known remote queue point" for each remote messaging engine
# UUID that we have remote put/get state for.
class KnownRemoteQueuePoint:
  "A class for printing a runtime summary of a known remote queue point"
  consumerTransmitter = None
  inboundReceiver = None
  def __init__ (self, qpMBean, uuid):
    self.uuid = uuid
    if mesByUUID.has_key(uuid): self.name = mesByUUID[uuid].name
    else: self.name = "Unknown"
    self.qpMBean = qpMBean
  # Print an XML tag summarising this KnownRemoteQueuePoint
  def printSummaryXML(self, indent):
    out.write(indent + '<KnownRemoteQueuePoint me="' + self.name + '" meUUID="' + self.uuid + '"')
    try:
      currentInboundMessages = 0
      inboundMessagesReceived = 0
      if self.inboundReceiver != None:
        currentInboundMessages = self.inboundReceiver.getDepth()
        inboundMessagesReceived = self.inboundReceiver.getNumberOfMessagesReceived()
      out.write(' currentInboundMessages="' + str(currentInboundMessages) + '"')
      out.write(' inboundMessagesReceived="' + str(inboundMessagesReceived) + '"')
      currentMessageRequests = 0
      completedMessageRequests = 0
      messageRequestsReceived = 0
      if self.consumerTransmitter != None:
        currentMessageRequests = self.consumerTransmitter.getDepth()
        completedMessageRequests = self.consumerTransmitter.getNumberOfCompletedRequests()
        messageRequestsReceived = self.consumerTransmitter.getNumberOfRequestsReceived()
      out.write(' currentMessageRequests="' + str(currentMessageRequests) + '"')
      out.write(' completedMessageRequests="' + str(completedMessageRequests) + '"')
      out.write(' messageRequestsReceived="' + str(messageRequestsReceived) + '"')
      # Only need one line for this
      out.write('/>\n')
    except:
      et.printErrorAndCloseTag(sys.exc_info(), 1)

# Script execution starts here ...

# The only input parameter is the name of the output file.
# Default to STDOUT if not specified
if len(sys.argv) > 0:
  print "Writing output to", sys.argv[0]
  out = open(sys.argv[0], 'w')
  fileOpened = 1
else:
  out = sys.stdout
  fileOpened = 0

# Lookup all MEs in the configuration and build a ME UUID to name dictionary
mesByUUID = {}
mesByBus = {}
mes = AdminConfig.list("SIBMessagingEngine").split("\n")
for me in mes:
  # Remove carriage returns on Windows
  me = me.strip()
  # Construct an ME object
  meBus  = AdminConfig.showAttribute(me, "busName")
  meName = AdminConfig.showAttribute(me, "name")
  meUUID = AdminConfig.showAttribute(me, "uuid")
  meObject = MessagingEngine(meBus, meName, meUUID)
  # Place the ME into the map by UUID
  mesByUUID[meUUID] = meObject
  # Place the ME into the list for this bus
  if (mesByBus.has_key(meBus)): meList = mesByBus[meBus]
  else: meList = []
  meList.append(meObject)
  mesByBus[meBus] = meList

# Iterate through the buses, and MEs within each bus
try:
  out.write('<?xml version="1.0"?>\n')
  out.write('<SIBusSummary>\n')
  for busName in mesByBus.keys():
    indent = ' '
    out.write(indent + '<Bus name="' + busName + '">\n')
```

```
    meList = mesByBus[busName]
    for me in meList:
      me.printRuntimeStateXML(indent + ' ')
    out.write(indent + '</Bus>\n')
  out.write('</SIBusSummary>\n')
finally:
  if (fileOpened): out.close()
```

The following example shows sample output for the script.

```
<?xml version="1.0"?>
<SIBusSummary>
 <Bus name="Bus2">
  <MessagingEngine name="Node1.server1-Bus2" uuid="B04215B7389FDA8F" state="Started" activeServer="server1">
   <QueuePoints>
    <QueuePoint name="Bus2Queue1@Node1.server1-Bus2" state="ACTIVE" depth="9" highMessageThreshold="50000"
 sendAllowed="1"/>
    <QueuePoint name="RemoteMediatedQueue1@Node1.server1-Bus2" state="ACTIVE" depth="5" highMessageThreshold="50000"
sendAllowed="1">
     <KnownRemoteQueuePoint me="Node1.server2-Bus2" meUUID="CC8EAD412746BA2A" currentInboundMessages="0"
inboundMessagesReceived="0" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
    <QueuePoint name="_PSIMP.PROXY.QUEUE_B04215B7389FDA8F@Node1.server1-Bus2" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1">
     <KnownRemoteQueuePoint me="Node1.server2-Bus2" meUUID="CC8EAD412746BA2A" currentInboundMessages="0"
 inboundMessagesReceived="2" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
    <QueuePoint name="_PTRM_B04215B7389FDA8F@Node1.server1-Bus2" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_SYSTEM.Exception.Destination.Node1.server1-Bus2@Node1.server1-Bus2" state="ACTIVE"
depth="3" highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.TDRECEIVER_B04215B7389FDA8F@Node1.server1-Bus2" state="ACTIVE" depth="0" h
ighMessageThreshold="50000" sendAllowed="1"/>
   </QueuePoints>
   <RemoteQueuePoints>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_CC8EAD412746BA2A@Node1.server1-Bus2"
remoteME="Node1.server2-Bus2" remoteMEUUID="CC8EAD412746BA2A" currentOutboundMessages="0"
 outboundMessageSent="4" currentMessageRequests="0" completedMessageRequests="0" messageRequestsIssued="0"/>
   </RemoteQueuePoints>
   <MediationPoints>
   </MediationPoints>
   <RemoteMediationPoints>
    <RemoteMediationPoint name="RemoteMediatedQueue1@Node1.server1-Bus2" remoteME="Node1.server2-Bus2"
remoteMEUUID="CC8EAD412746BA2A" currentOutboundMessages="0" outboundMessageSent="0"/>
   </RemoteMediationPoints>
   <PublicationPoints>
    <PublicationPoint name="Default.Topic.Space@Node1.server1-Bus2" depth="0" highMessageThreshold="50000"
sendAllowed="1">
    </PublicationPoint>
   </PublicationPoints>
   <BusLinks>
    <BusLink name="Bus1Bus2Link" foreignBus="Bus1" state="STARTED">
     <LinkReceiver state="STARTED" me="Node1.server1-Bus1" meUUID="92FF69453638CD2F" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="4" timeSinceLastMessageReceived="18809ms"   />
     <LinkReceiver state="STARTED" me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="0"   />
     <LinkReceiver state="STARTED" me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" receiverType="PUBLICATION"
 topicSpace="" currentInboundMessages="0" messagesReceived="0"   />
     <LinkReceiver state="STARTED" me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="2" timeSinceLastMessageReceived="20091ms"   />
     <LinkReceiver state="STARTED" me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="0"   />
     <LinkTransmitter state="STARTED" linkType="SIBVirtualGatewayLink" transmitterType="QUEUE" sendAllowed="1"
currentOutboundMessages="0" messagesSent="3" timeSinceLastMessageSent="17509ms"/>
    </BusLink>
   </BusLinks>
   <WMQLinks>
    <WMQLink name="MQBus1Link" state="RUNNING" messagesReceived="0">
     <SenderChannel channelName="TO.PAB" state="STANDBY" virtualQmgr="WAS80" ipAddress="" messagesSent="0"
 currentLUWID="0" currentSequenceNo="0" inDoubt="0">
      <SenderChannelTransmitter status="STARTED" currentOutboundMessages="0" messagesSent="0">
       <KnownRemoteSenderChannelTransmitter me="Node1.server2-Bus2" meUUID="CC8EAD412746BA2A"
currentInboundMessages="0" inboundMessagesReceived="0"/>
      </SenderChannelTransmitter>
     </SenderChannel>
     <ReceiverChannel state="INACTIVE">
     </ReceiverChannel>
    </WMQLink>
   </WMQLinks>
  </MessagingEngine>
  <MessagingEngine name="Node1.server2-Bus2" uuid="CC8EAD412746BA2A" state="Started" activeServer="server2">
   <QueuePoints>
    <QueuePoint name="_SYSTEM.Exception.Destination.Node1.server2-Bus2@Node1.server2-Bus2" state="ACTIVE"
 depth="0" highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.PROXY.QUEUE_CC8EAD412746BA2A@Node1.server2-Bus2" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1">
     <KnownRemoteQueuePoint me="Node1.server1-Bus2" meUUID="B04215B7389FDA8F" currentInboundMessages="0"
inboundMessagesReceived="12" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
```

```
    <QueuePoint name="_PTRM_CC8EAD412746BA2A@Node1.server2-Bus2" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.TDRECEIVER_CC8EAD412746BA2A@Node1.server2-Bus2" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
   </QueuePoints>
   <RemoteQueuePoints>
    <RemoteQueuePoint name="RemoteMediatedQueue1@Node1.server2-Bus2" remoteME="Node1.server1-Bus2"
remoteMEUUID="B04215B7389FDA8F" currentOutboundMessages="0" outboundMessageSent="2" currentMessageRequests="0"
 completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_B04215B7389FDA8F@Node1.server2-Bus2" remoteME="Node1.server1-Bus2"
 remoteMEUUID="B04215B7389FDA8F" currentOutboundMessages="0" outboundMessageSent="8" currentMessageRequests="0"
 completedMessageRequests="0" messageRequestsIssued="0"/>
   </RemoteQueuePoints>
   <MediationPoints>
    <MediationPoint name="RemoteMediatedQueue1@Node1.server2-Bus2" status="Started" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
   </MediationPoints>
   <RemoteMediationPoints>
   </RemoteMediationPoints>
   <PublicationPoints>
    <PublicationPoint name="Default.Topic.Space@Node1.server2-Bus2" depth="0" highMessageThreshold="50000"
 sendAllowed="1">
    </PublicationPoint>
   </PublicationPoints>
   <BusLinks>
    <BusLink name="Bus2:MQBus1" foreignBus="MQBus1">
     <LinkTransmitter state="STARTED" linkType="SIBVirtualMQLink" transmitterType="QUEUE" sendAllowed="1"
 currentOutboundMessages="0" messagesSent="0"/>
    </BusLink>
   </BusLinks>
   <WMQLinks>
   </WMQLinks>
  </MessagingEngine>
 </Bus>
 <Bus name="Bus1">
  <MessagingEngine name="Node1.server1-Bus1" uuid="92FF69453638CD2F" state="Started" activeServer="server1">
   <QueuePoints>
    <QueuePoint name="_PTRM_92FF69453638CD2F@Node1.server1-Bus1" state="ACTIVE" depth="0" highMessageThreshold="50000"
sendAllowed="1"/>
    <QueuePoint name="_SYSTEM.Exception.Destination.Node1.server1-Bus1@Node1.server1-Bus1" state="ACTIVE" depth="0"
 highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.PROXY.QUEUE_92FF69453638CD2F@Node1.server1-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1">
     <KnownRemoteQueuePoint me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" currentInboundMessages="0"
inboundMessagesReceived="3" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
     <KnownRemoteQueuePoint me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" currentInboundMessages="0"
inboundMessagesReceived="3" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
    <QueuePoint name="_PSIMP.TDRECEIVER_92FF69453638CD2F@Node1.server1-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="Bus1Queue1@Node1.server1-Bus1" state="ACTIVE" depth="3" highMessageThreshold="50000"
sendAllowed="1">
     <KnownRemoteQueuePoint me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" currentInboundMessages="0"
inboundMessagesReceived="0" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
     <KnownRemoteQueuePoint me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" currentInboundMessages="0"
inboundMessagesReceived="0" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
   </QueuePoints>
   <RemoteQueuePoints>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_122AAD73434FF5DA@Node1.server1-Bus1" remoteME="cluster1.001-Bus1"
remoteMEUUID="122AAD73434FF5DA" currentOutboundMessages="0" outboundMessageSent="2" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_C96051A1F0F91AB3@Node1.server1-Bus1" remoteME="cluster1.000-Bus1"
remoteMEUUID="C96051A1F0F91AB3" currentOutboundMessages="0" outboundMessageSent="2" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
   </RemoteQueuePoints>
   <MediationPoints>
   </MediationPoints>
   <RemoteMediationPoints>
   </RemoteMediationPoints>
   <PublicationPoints>
    <PublicationPoint name="Default.Topic.Space@Node1.server1-Bus1" depth="10" highMessageThreshold="50000"
 sendAllowed="1">
     <RemotePublicationPoint me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" currentInboundMessages="0"
inboundMessagesReceived="0"/>
     <RemotePublicationPoint me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" currentInboundMessages="0"
 inboundMessagesReceived="0"/>
     <Subscription subscriberId="MySubName1" depth="10" topics=""/>
     <Subscription subscriberId="MySubName2" depth="5" topics=""/>
    </PublicationPoint>
   </PublicationPoints>
   <BusLinks>
    <BusLink name="Bus1Bus2Link" foreignBus="Bus2" state="STARTED">
     <LinkReceiver state="STARTED" me="Node1.server1-Bus2" meUUID="B04215B7389FDA8F" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="3" timeSinceLastMessageReceived="19311ms"   />
     <LinkReceiver state="STARTED" me="Node1.server1-Bus2" meUUID="B04215B7389FDA8F" receiverType="PUBLICATION"
topicSpace="" currentInboundMessages="0" messagesReceived="0"   />
     <LinkTransmitter state="STARTED" linkType="SIBVirtualGatewayLink" transmitterType="QUEUE" sendAllowed="1"
currentOutboundMessages="0" messagesSent="4" timeSinceLastMessageSent="20435ms"/>
    </BusLink>
```

```
   </BusLinks>
   <WMQLinks>
   </WMQLinks>
  </MessagingEngine>
  <MessagingEngine name="cluster1.000-Bus1" uuid="C96051A1F0F91AB3" state="Started" activeServer="clusServer1">
   <QueuePoints>
    <QueuePoint name="_PSIMP.TDRECEIVER_C96051A1F0F91AB3@cluster1.000-Bus1" state="ACTIVE" depth="0"
 highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_SYSTEM.Exception.Destination.cluster1.000-Bus1@cluster1.000-Bus1" state="ACTIVE"
depth="0" highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PTRM_C96051A1F0F91AB3@cluster1.000-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.PROXY.QUEUE_C96051A1F0F91AB3@cluster1.000-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1">
     <KnownRemoteQueuePoint me="cluster1.001-Bus1" meUUID="122AAD73434FF5DA" currentInboundMessages="0"
inboundMessagesReceived="4" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
     <KnownRemoteQueuePoint me="Node1.server1-Bus1" meUUID="92FF69453638CD2F" currentInboundMessages="0"
inboundMessagesReceived="11" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
    <QueuePoint name="Bus2WLMQueue1@cluster1.000-Bus1" state="ACTIVE" depth="0" highMessageThreshold="50000"
 sendAllowed="1"/>
   </QueuePoints>
   <RemoteQueuePoints>
    <RemoteQueuePoint name="Bus1Queue1@cluster1.000-Bus1" remoteME="Node1.server1-Bus1"
remoteMEUUID="92FF69453638CD2F" currentOutboundMessages="0" outboundMessageSent="1"
currentMessageRequests="0" completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_122AAD73434FF5DA@cluster1.000-Bus1" remoteME="cluster1.001-Bus1"
remoteMEUUID="122AAD73434FF5DA" currentOutboundMessages="0" outboundMessageSent="4" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_92FF69453638CD2F@cluster1.000-Bus1" remoteME="Node1.server1-Bus1"
remoteMEUUID="92FF69453638CD2F" currentOutboundMessages="0" outboundMessageSent="11" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
   </RemoteQueuePoints>
   <MediationPoints>
   </MediationPoints>
   <RemoteMediationPoints>
   </RemoteMediationPoints>
   <PublicationPoints>
    <PublicationPoint name="Default.Topic.Space@cluster1.000-Bus1" depth="0" highMessageThreshold="50000"
sendAllowed="1">
     <Subscription subscriberId="MySubName3" depth="0" topics=""/>
    </PublicationPoint>
   </PublicationPoints>
   <BusLinks>
    <BusLink name="Bus1:Bus2" foreignBus="Bus2">
     <LinkTransmitter state="STOPPED" linkType="SIBVirtualGatewayLink" transmitterType="QUEUE" sendAllowed="1"
 currentOutboundMessages="0" messagesSent="1" timeSinceLastMessageSent="3471037ms"/>
    </BusLink>
   </BusLinks>
   <WMQLinks>
   </WMQLinks>
  </MessagingEngine>
  <MessagingEngine name="cluster1.001-Bus1" uuid="122AAD73434FF5DA" state="Started" activeServer="clusServer2">
   <QueuePoints>
    <QueuePoint name="_PSIMP.TDRECEIVER_122AAD73434FF5DA@cluster1.001-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="_PSIMP.PROXY.QUEUE_122AAD73434FF5DA@cluster1.001-Bus1" state="ACTIVE" depth="0"
 highMessageThreshold="50000" sendAllowed="1">
     <KnownRemoteQueuePoint me="Node1.server1-Bus1" meUUID="92FF69453638CD2F" currentInboundMessages="0"
 inboundMessagesReceived="12" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
     <KnownRemoteQueuePoint me="cluster1.000-Bus1" meUUID="C96051A1F0F91AB3" currentInboundMessages="0"
inboundMessagesReceived="2" currentMessageRequests="0" completedMessageRequests="0" messageRequestsReceived="0"/>
    </QueuePoint>
    <QueuePoint name="_SYSTEM.Exception.Destination.cluster1.001-Bus1@cluster1.001-Bus1" state="ACTIVE"
 depth="0" highMessageThreshold="50000" sendAllowed="1"/>
    <QueuePoint name="Bus2WLMQueue1@cluster1.001-Bus1" state="ACTIVE" depth="0" highMessageThreshold="50000"
sendAllowed="1"/>
    <QueuePoint name="_PTRM_122AAD73434FF5DA@cluster1.001-Bus1" state="ACTIVE" depth="0"
highMessageThreshold="50000" sendAllowed="1"/>
   </QueuePoints>
   <RemoteQueuePoints>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_92FF69453638CD2F@cluster1.001-Bus1"
remoteME="Node1.server1-Bus1" remoteMEUUID="92FF69453638CD2F" currentOutboundMessages="0"
outboundMessageSent="12" currentMessageRequests="0" completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="Bus1Queue1@cluster1.001-Bus1" remoteME="Node1.server1-Bus1"
remoteMEUUID="92FF69453638CD2F" currentOutboundMessages="0" outboundMessageSent="1" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
    <RemoteQueuePoint name="_PSIMP.PROXY.QUEUE_C96051A1F0F91AB3@cluster1.001-Bus1" remoteME="cluster1.000-Bus1"
remoteMEUUID="C96051A1F0F91AB3" currentOutboundMessages="0" outboundMessageSent="2" currentMessageRequests="0"
completedMessageRequests="0" messageRequestsIssued="0"/>
   </RemoteQueuePoints>
   <MediationPoints>
   </MediationPoints>
   <RemoteMediationPoints>
   </RemoteMediationPoints>
   <PublicationPoints>
    <PublicationPoint name="Default.Topic.Space@cluster1.001-Bus1" depth="0" highMessageThreshold="50000"
 sendAllowed="1">
     <Subscription subscriberId="MySubName4" depth="0" topics=""/>
    </PublicationPoint>
```

```
   </PublicationPoints>
   <BusLinks>
    <BusLink name="Bus1:Bus2" foreignBus="Bus2">
     <LinkTransmitter state="STOPPED" linkType="SIBVirtualGatewayLink" transmitterType="QUEUE" sendAllowed="1"
 currentOutboundMessages="0" messagesSent="1" timeSinceLastMessageSent="3471168ms"/>
    </BusLink>
   </BusLinks>
   <WMQLinks>
   </WMQLinks>
  </MessagingEngine>
 </Bus>
</SIBusSummary>
```

Chapter 8. Scripting web applications

This page provides a starting point for finding information about web applications, which are comprised of one or more related files that you can manage as a unit, including HTML files; Servlets can support dynamic web page content, provide database access, serve multiple clients at one time, and filter data; and Java ServerPages (JSP) files enable the separation of the HTML code from the business logic in web pages. IBM extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming.

Configuring applications for session management using scripting

This task provides an example that uses the AdminConfig object to configure a session manager for the application.

Before you begin

An application must be installed on a running server.

About this task

You can use the AdminConfig object to set configurations in an application. Some configuration settings are not available through the AdminConfig object.

Procedure

1. Start the wsadmin scripting tool.
2. Identify the deployment configuration object for the application and assign it to the deployment variable.

   Note: This step is not needed for an OSGi application. See Adding an EBA asset to a composition unit using wsadmin commands and Modifying the configuration of an EBA composition unit using wsadmin commands.

   For example:

   - Using Jacl:

```
set deployments [$AdminConfig getid /Deployment:myApp/]
```

   - Using Jython:

```
deployments = AdminConfig.getid('/Deployment:myApp/')
print deployments
```

   where:

Table 72. getid command elements. Run the getid command to identify a deployment object.

`set`	is a Jacl command
`deployments`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Deployment`	is an attribute
myApp	is the value of the attribute

   Example output:

```
myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
```

3. Retrieve the application deployment object and assign it to the appDeploy variable. For example:

- Using Jacl:

```
set appDeploy [$AdminConfig showAttribute $deployments deployedObject]
```

- Using Jython:

```
appDeploy = AdminConfig.showAttribute(deployments, 'deployedObject')
print appDeploy
```

Note: For an OSGi application, use the following jython code for this step:

```
appDeploy = AdminTask.getOSGiApplicationDeployedObject('-cuName cu_name')
```

where:

Table 73. set command elements. Run the set command to assign the deployment object a value.

`set`	is a Jacl command
`appDeploy`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object that represents the WebSphere Application Server configuration
`showAttribute`	is an AdminConfig command
`deployments`	evaluates the ID of the deployment object that is specified in step number 1
`deployedObject`	is an attribute
cu_name	is the name of the composition unit

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#ApplicationDeployment_1)
```

4. To obtain a list of attributes that you can set for a session manager, use the **attributes** command. For example:

- Using Jacl:

```
$AdminConfig attributes SessionManager
```

- Using Jython:

```
print AdminConfig.attributes('SessionManager')
```

where:

Table 74. attributes command elements. Run the attributes command to list attributes of a session manager.

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object that represents the WebSphere Application Server configuration
`attributes`	is an AdminConfig command
`SessionManager`	is an attribute

Example output:

```
"accessSessionOnTimeout Boolean"
"allowSerializedSessionAccess Boolean"
"context ServiceContext@"
"defaultCookieSettings Cookie"
"enable Boolean"
"enableCookies Boolean"
"enableProtocolSwitchRewriting Boolean"
"enableSSLTracking Boolean"
"enableSecurityIntegration Boolean"
"enableUrlRewriting Boolean"
"maxWaitTime Integer"
"properties Property(TypedProperty)*"
```

```
"sessionDRSPersistence DRSSettings"
"sessionDatabasePersistence SessionDatabasePersistence"
"sessionPersistenceMode ENUM(DATABASE, DATA_REPLICATION, NONE)"
"tuningParams TuningParams"
```

When you configure an application for session management, it is recommended that you specify each attribute.

gotcha: If you are setting up the session management attributes for a cluster, you must also update the targetMappings element of the AdminConfig object for the cluster before the settings you specify for the sessionManagment element become effective. If you do not update the targetMappings element, the settings are not effective even though they appear in the deployment.xml file.

5. Set up the attributes for the session manager.

   The following example sets four top-level attributes in the session manager. You can modify the example to set other attributes of the session manager, including the nested attributes in DRSSettings, SessionDataPersistence, and TuningParms object types.

   gotcha: The session manager requires that you set both the defaultCookieSettings and tuningParams attributes before you initialize an application. If you do not set these attributes, the session manager cannot initialize the application, and the application does not start.

   To list the attributes for those object types, use the **attributes** command of the AdminConfig object.

   - Using Jacl:

```
set attr1 [list enableSecurityIntegration true]
set attr2 [list maxWaitTime 30]
set attr3 [list sessionPersistenceMode NONE]
set kuki [list maximumAge -1]
set cookie [list $kuki]
Set cookieSettings [list defaultCookieSettings $cookie]
set attrs [list $attr1 $attr2 $attr3 $cookieSettings]
set sessionMgr [list sessionManagement $attrs]
```

   Example output using Jacl:

```
sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30} {sessionPersistenceMode NONE}
{defaultCookieSettings {{maximumAge -1}}}}
```

   - Using Jython:

```
attr1 = ['enableSecurityIntegration', 'true']
attr2 = ['maxWaitTime', 30]
attr3 = ['sessionPersistenceMode', 'NONE']
kuki = ['maximumAge', -1]
cookie = [kuki]
cookieSettings = ['defaultCookieSettings', cookie]
attrs = [attr1, attr2, attr3, cookieSettings]
sessionMgr = [['sessionManagement', attrs]]
```

   Example output using Jython:

```
[[sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30], [sessionPersistenceMode, NONE],
[defaultCookieSettings [[maximumAge, -1]]]]
```

   where:

Table 75. set command elements. Run the set command to set attributes for a session manager.

`set`	is a Jacl command
`attr1`, `attr2`, `attr3`, `attrs`, `sessionMgr`	are variable names
`$`	is a Jacl operator for substituting a variable name with its value
`enableSecurityIntegration`	is an attribute
`true`	is a value of the enableSecurityIntegration attribute
`maxWaitTime`	is an attribute
`30`	is a value of the maxWaitTime attribute
`sessionPersistenceMode`	is an attribute

Table 75. set command elements (continued). Run the set command to set attributes for a session manager.

NONE	is a value of the sessionPersistenceMode attribute

6. Perform one of the following:
   - Create the session manager for the application. For example:
     – Using Jacl:

```
$AdminConfig create ApplicationConfig $appDeploy [list $sessionMgr]
```

     – Using Jython:

```
print AdminConfig.create('ApplicationConfig', appDeploy, sessionMgr)
```

     where:

Table 76. create command elements. Run the create command to create a session manager.

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object that represents the WebSphere Application Server configuration
create	is an AdminConfig command
ApplicationConfig	is an attribute
appDeploy	evaluates the ID of the deployed application that is specified in step number 2
list	is a Jacl command
sessionMgr	evaluates the ID of the session manager that is specified in step number 4

     Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#ApplicationConfig_1)
```

   - If a session manager already exists, use the **modify** command of the AdminConfig object to update the configuration of the session manager. For example:
     – Using Jacl:

```
set configs [lindex [$AdminConfig showAttribute $appDeploy configs] 0]
set appConfig [lindex $configs 0]
set SM [$AdminConfig showAttribute $appConfig sessionManagement]
$AdminConfig modify $SM $attrs
```

     – Using Jython:

```
configs = AdminConfig.showAttribute (appDeploy, 'configs')
appConfig = configs[1:len(configs)-1]
SM = AdminConfig.showAttribute (appConfig, 'sessionManagement')
AdminConfig.modify (SM, attrs)
```

7. Save the configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

8. Synchronize the node.

   Use the syncActiveNode or syncNode scripts in the AdminNodeManagement script library to propagate the configuration changes to node or nodes.
   - Use the syncActiveNodes script to propagate the changes to each node in the cell, as the following example demonstrates:

```
AdminNodeManagement.syncActiveNodes()
```

   - Use the syncNode script to propagate the changes to a specific node, as the following example demonstrates:

```
AdminNodeManagement.syncNode("myNode")
```

Configuring applications for session management in web modules using scripting

Use scripting and the wsadmin tool to configure applications for session management in web modules.

About this task

You can use the AdminApp object to set configurations in an application. Some configuration settings are not available through the AdminApp object. The following task uses the AdminConfig object to configure a session manager for a web module in the application.

Procedure

1. Start the wsadmin scripting tool.
2. Identify the deployment configuration object for the application and assign it to the deployment variable. For example:
   - Using Jacl:

```
set deployments [$AdminConfig getid /Deployment:myApp/]
```

   - Using Jython:

```
deployments = AdminConfig.getid('/Deployment:myApp/')
print deployments
```

   where:

Table 77. Deployment configuration values. The following table describes the elements of the getid command.

`set`	is a Jacl command
`deployments`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object that represents the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Deployment`	is an attribute
myApp	is the value of the attribute

   Example output:

```
myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
```

3. Get all the modules in the application and assign them to the modules variable. For example:
   - Using Jacl:

```
set appDeploy [$AdminConfig showAttribute $deployments deployedObject]
set mod1 [$AdminConfig showAttribute $appDeploy modules]
set mod1 [lindex $mod1 0]
```

     Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#WebModuleDeployment_1)
(cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#EJBModuleDeployment_1)
(cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#WebModuleDeployment_2)
```

   - Using Jython:

```
appDeploy = AdminConfig.showAttribute(deployments, 'deployedObject')
mod1 = AdminConfig.showAttribute(appDeploy, 'modules')
print mod1
```

     Example output:

```
[(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#WebModuleDeployment_1)
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#EJBModuleDeployment_1)
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#EJBModuleDeployment_2)]
```

   where:

Table 78. Application module values. The following table describes the elements of the showAttribute AdminConfig command.

set	is a Jacl command
appDeploy	is a variable name
mod1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object that represents the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
deployments	evaluates the ID of the deployment object that is specified in step number 1
deployedObject	is an attribute

4. To obtain a list of attributes that you can set for a session manager, use the **attributes** command. For example:
   - Using Jacl:

```
$AdminConfig attributes SessionManager
```

   - Using Jython:

```
print AdminConfig.attributes('SessionManager')
```

   where:

Table 79. attributes AdminConfig command. The following table describes the elements for the attributes AdminConfig command.

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object that represents the WebSphere Application Server configuration
attributes	is an AdminConfig command
SessionManager	is an attribute

   Example output:

```
"accessSessionOnTimeout Boolean"
"allowSerializedSessionAccess Boolean"
"context ServiceContext@"
"defaultCookieSettings Cookie"
"enable Boolean"
"enableCookies Boolean"
"enableProtocolSwitchRewriting Boolean"
"enableSSLTracking Boolean"
"enableSecurityIntegration Boolean"
"enableUrlRewriting Boolean"
"maxWaitTime Integer"
"properties Property(TypedProperty)*"
"sessionDRSPersistence DRSSettings"
"sessionDatabasePersistence SessionDatabasePersistence"
"sessionPersistenceMode ENUM(DATABASE, DATA_REPLICATION, NONE)"
"tuningParams TuningParams"
```

5. Set up the attributes for session manager. The following example sets four top-level attributes in the session manager.

   You can modify the example to set other attributes in the session manager, including the nested attributes in Cookie, DRSSettings, SessionDataPersistence, and TuningParms object types. To list the attributes for those object types, use the **attributes** command of AdminConfig object.

gotcha: The session manager requires that you set both the defaultCookieSettings and tuningParams attributes before you initialize an application. If you do not set these attributes, the session manager cannot initialize the application, and the application does not start.

- Using Jacl:

```
set attr0 [list enable true]
set attr1 [list enableSecurityIntegration true]
set attr2 [list maxWaitTime 30]
set attr3 [list sessionPersistenceMode NONE]
set attr4 [list enableCookies true]
set attr5 [list invalidationTimeout 45]
set tuningParmsDetailList [list $attr5]
set tuningParamsList [list tuningParams $tuningParmsDetailList]
set pwdList [list password 95ee608]
set userList [list userId Administrator]
set dsNameList [list datasourceJNDIName jdbc/session]
set dbPersistenceList [list $dsNameList $userList $pwdList]
set sessionDBPersistenceList [list $dbPersistenceList]
set sessionDBPersistenceList [list sessionDatabasePersistence $dbPersistenceList]
set kuki [list maximumAge 1000]
set cookie [list $kuki]
set cookieSettings [list defaultCookieSettings $cookie]
set sessionManagerDetailList [list $attr0 $attr1 $attr2 $attr3 $attr4 $cookieSettings
$tuningParamsList $sessionDBPersistenceList]
set sessionMgr [list sessionManagement $sessionManagerDetailList]
set id [$AdminConfig create ApplicationConfig $appDeploy [list $sessionMgr] configs]
set targetMappings [lindex [$AdminConfig showAttribute $appDeploy targetMappings] 0]
set attrs [list config $id]
$AdminConfig modify $targetMappings [list $attrs]
```

Note: The last five lines in the sample above assume that you deployed the web module to only one target server. You can target a module to multiple servers or clusters, by using a loop, if you want to apply the update to each target. Replace the last six lines of the sample above with the following code to apply updates to multiple targets:

```
set id [$AdminConfig create ApplicationConfig $appDeploy [list $sessionMgr] configs]

set targetMappings [lindex [$AdminConfig showAttribute $appDeploy targetMappings] 0]

foreach target $targetMappings {
    if {[regexp DeploymentTargetMapping $target] == 1} {
        set attrs [list config $id]
        $AdminConfig modify $target [list $attrs]
    }
}
```

Example output using Jacl:

```
sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30}
{sessionPersistenceMode NONE} {enabled true}}
```

- Using Jython:

```
attr0 = ['enable', 'true']
attr1 = ['enableSecurityIntegration', 'true']
attr2 = ['maxWaitTime', 30]
attr3 = ['sessionPersistenceMode', 'NONE']
attr4 = ['enableCookies', 'true']
attr5 = ['invalidationTimeout', 45]
tuningParmsDetailList = [attr5]
tuningParamsList = ['tuningParams', tuningParmsDetailList]
pwdList = ['password', '95ee608']
userList = ['userId', 'Administrator']
dsNameList = ['datasourceJNDIName', 'jdbc/session']
dbPersistenceList = [dsNameList, userList, pwdList]
sessionDBPersistenceList = [dbPersistenceList]
sessionDBPersistenceList = ['sessionDatabasePersistence', dbPersistenceList]
kuki = ['maximumAge', 1000]
cookie = [kuki]
cookieSettings = ['defaultCookieSettings', cookie]
sessionManagerDetailList = [attr0, attr1, attr2, attr3, attr4, cookieSettings,
tuningParamsList, sessionDBPersistenceList]
sessionMgr = ['sessionManagement', sessionManagerDetailList]
id = AdminConfig.create('ApplicationConfig', appDeploy,[sessionMgr], 'configs')
targetMappings = AdminConfig.showAttribute(appDeploy, 'targetMappings')
```

```
targetMappings = targetMappings[1:len(targetMappings)-1]
print targetMappings
attrs = ['config', id]
AdminConfig.modify(targetMappings,[attrs])
```

Note: The last six lines in the sample above assume that you deployed the web module to only one target server. You can target a module to multiple servers or clusters, by using a loop, if you want to apply the update to each target. Replace the last six lines of the sample above with the following code to apply updates to multiple targets:

```
id = AdminConfig.create('ApplicationConfig', appDeploy,[sessionMgr], 'configs')

targetMappings = AdminConfig.showAttribute(appDeploy, 'targetMappings')
targetMappings = targetMappings[1:len(targetMappings)-1].split(" ")
for target in targetMappings:
 if target.find('DeploymentTargetMapping') != -1:
  attrs = ['config', id]
  AdminConfig.modify(target,[attrs])
 #endif
#endfor
```

Example output using Jython:

```
[sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30], [sessionPersistenceMode, NONE]]
```

6. Set up the attributes for the web module. For example:
   - Using Jacl:

```
set nameAttr [list name myWebModuleConfig]
set descAttr [list description "Web Module config post create"]
set webAttrs [list $nameAttr $descAttr $sessionMgr]
```

Example output:

```
{name myWebModuleConfig} {description {Web Module config post create}}
{sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30}
{sessionPersistenceMode NONE} {enabled true}}}
```

   - Using Jython:

```
nameAttr = ['name', 'myWebModuleConfig']
descAttr = ['description', "Web Module config post create"]
webAttrs = [nameAttr, descAttr, sessionMgr]
```

Example output:

```
[[name, myWebModuleConfig], [description, "Web Module config post create"],
[sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30],
[sessionPersistenceMode, NONE], [enabled, true]]]]
```

where:

Table 80. Setting the attributes for the web module. The following table describes the elements for setting the attributes of the web module.

set	is a Jacl command
nameAttr, descAttr, webAttrs	are variable names
$	is a Jacl operator for substituting a variable name with its value
name	is an attribute
myWebModuleConfig	is a value of the name attribute
description	is an attribute
Web Module config post create	is a value of the description attribute

7. Create the session manager for each web module in the application. You can modify the following example to set other attributes of the session manager in a web module configuration. You must also define a target mapping for this step.
   - Using Jacl:

```
foreach module $mod1 {
 if {[regexp WebModuleDeployment $module] == 1} {
  set moduleConfig [$AdminConfig create WebModuleConfig $module $webAttrs]
  set targetMappings [lindex [$AdminConfig showAttribute $module targetMappings] 0]
  set attrs [list config $moduleConfig]
  $AdminConfig modify $targetMappings [list $attrs]
 }
}
```

Note: You can add an optional, additional loop to assign new web module configuration to each target, if the web module is deployed to more than one target server:

```
foreach module $mod1 {
    if {[regexp WebModuleDeployment $module] == 1} {
        set moduleConfig [$AdminConfig create WebModuleConfig $module $webAttrs]
        set targetMappings [lindex [$AdminConfig showAttribute $module targetMappings] 0]
        foreach target $targetMappings {
        if {[regexp DeploymentTargetMapping $target] == 1} {
            set attrs [list config $moduleConfig]
            $AdminConfig modify $target [list $attrs]
        }
    }
}
```

- Using Jython:

```
arrayModules = mod1[1:len(mod1)-1].split(" ")
for module in arrayModules:
 if module.find('WebModuleDeployment') != -1:
  AdminConfig.create('WebModuleConfig', module, webAttrs)
  targetMappings = targetMappings[1:len(targetMappings)-1]
  attrs = ['config', moduleConfig]
  AdminConfig.modify (targetMappings, [attrs])
```

Note: You can add an optional, additional loop to assign new web module configuration to each target, if the web module is deployed to more than one target server:

```
arrayModules = mod1[1:len(mod1)-1].split(" ")
for module in arrayModules:
  if module.find('WebModuleDeployment') != -1:
    moduleConfig = AdminConfig.create('WebModuleConfig', module, webAttrs)
    attrs = ['config', moduleConfig]
    targetMappings = AdminConfig.showAttribute(appDeploy, 'targetMappings')
    targetMappings = targetMappings[1:len(targetMappings)-1].split(" ")
    for target in targetMappings:
        if target.find('DeploymentTargetMapping') != -1:
            attrs = ['config', moduleConfig]
            AdminConfig.modify(target,[attrs])
```

Example output:

```
myWebModuleConfig(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#WebModuleConfiguration_1)
```

If you do not specify the tuningParamsList attribute when you create the session manager, you will receive an error when you start the deployed application.

8. Save the configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

9. In a network deployment environment only, synchronize the node.

   Use the syncActiveNode or syncNode scripts in the AdminNodeManagement script library to propagate the configuration changes to node or nodes.

   - Use the syncActiveNodes script to propagate the changes to each node in the cell, as the following example demonstrates:

     ```
     AdminNodeManagement.syncActiveNodes()
     ```

   - Use the syncNode script to propagate the changes to a specific node, as the following example demonstrates:

     ```
     AdminNodeManagement.syncNode("myNode")
     ```

Chapter 9. Scripting for web services

This page provides a starting point for finding information about web services.

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Starting the wsadmin scripting client using wsadmin scripting

You can use the wsadmin tool to configure and administer application servers, application deployment, and server runtime operations.

About this task

The wsadmin tool provides the ability to automate configuration tasks for your environment by running scripts. However, there are some limitations for using the wsadmin tool, including:

- The wsadmin tool only supports the Jython and Jacl scripting languages.

  The Version 6.1 release of WebSphere Application Server represented the start of the deprecation process for the Jacl syntax that is associated with the wsadmin tool. The Jacl syntax for the wsadmin tool continues to remain in the product and is supported for at least two major product releases. After that time, the Jacl language support might be removed from the wsadmin tool. The Jython syntax for the wsadmin tool is the strategic direction for WebSphere Application Server administrative automation. The application server provides significantly enhanced administrative functions and tooling that support product automation and the use of the Jython syntax.

  gotcha: Not all of the WebSphere Application Server component classes are packaged in the same `.jar` file. If you are going to be using the wsadmin tool to run Jython scripts, include the jython.package.path system property on your wsadmin command to ensure that all of the required JAR files are set to the jython package path during wsadmin startup.

  ```
  ./wsadmin.sh -lang jython -javaoption
  "-Djython.package.path=/usr/WebSphere70/AppServer/plugins/com.ibm.ws.wlm.jar"
  ```

  If you want to invoke WebSphere Application Server functions from different WebSphere Application Server classes that are packaged in `.jar` files other than `runtime.jar` and `admin.jar`, you can include multiple jar files in the path specified for the jython.package.path system property, and separate them with semicolon (;).

  ```
  ./wsadmin.sh -lang jython -javaoption
  "-Djython.package.path=/usr/WebSphere70/AppServer/plugins/com.ibm.ws.wlm.jar;com.ibm.ws.wccm.jar"
  ```

  If you want to invoke WebSphere Application Server functions in a jython script using ws_ant, you can create a `.prop` file text file, and include the following line in this file:

  ```
  jython.package.path=/usr/WebSphere70/AppServer/plugins/com.ibm.ws.wlm.jar
  ```

  Then include the property file in the ant script xml file. For example:

  ```
  <taskdef name="wsadmin" classname="com.ibm.websphere.ant.tasks.WsAdmin"/>
  <target name="main" >
      <wsadmin conntype="NONE" lang="jython" failonerror="true" properties="/tmp/jython.prop"
               script="/home/fsgapp/MSTWasBuild/project/scripts/socr/socr/jython/configure.py">
      </wsadmin>
  </target>
  ```

- The wsadmin tool manages the installation, configuration, deployment, and runtime operations for application servers, deployment managers, administrative agents, and job managers that run the same version or a higher version of the product. The wsadmin tool cannot connect to an application server, deployment manager, administrative agent, or job manager that runs a product version which is older

than the version of the wsadmin tool. For example, a Version 6.x wsadmin client cannot connect to a Version 5.x application server. However, a Version 5.x wsadmin client can connect to a Version 6.x application server. This limitation exists because new functionality is added to the wsadmin tool in each product release. You cannot use new command functionality on application servers running previous product versions.

- The wsadmin tool operates at the deployment manager node level in a mixed-cell environment. Do not run wsadmin at the application server node level to ensure that all command functionality is available.

In a flexible management environment, you can connect the wsadmin tool to a base application server, deployment manager, administrative agent, or job manager process. If you do not specify the port of the base application server or the profile name assigned to the job manager, the wsadmin tool automatically connects to the administrative agent.

Procedure

1. Locate the command that starts the wsadmin scripting client.

   Choose one of the following:

   - Invoke the scripting process using a specific profile. The QShell command for invoking a scripting process is located in the *profile_root*/bin directory. The name of the QShell script is wsadmin. If you use this option, you do not need to specify the -profileName *profilename* parameter.
   - Invoke the scripting process using the default profile. The wsadmin Qshell command is located in the *app_server_root*/bin directory. If you do not want to connect to the default profile, you must specify the -profileName *profilename* parameter to indicate the profile that you want to use.

2. In a flexible management environment, determine whether to connect to a base application server, administrative agent, or job manager process.

   - Connect to the administrative agent process.

     Connect the wsadmin tool to the administrative agent to configure, manage, and administer servers. If you do not specify connection options, the wsadmin tool automatically connects to the administrative agent process. Use the following command to connect to the administrative agent:

     ```
     wsadmin -lang jython
     ```

   - Connect to a base application server process.

     Connect the wsadmin tool to a base application server to manage settings for a specific server of interest. Use this connection type when connecting to a node that contains one server and is registered with the administrative agent. Use the following command to connect to a base application server:

     ```
     wsadmin -conntype SOAP [-port 4213] -lang jython
     ```

   - Connect to the job manager process.

     Connect the wsadmin tool to the job manager to submit, monitor, and manage administrative jobs. Use the following command to connect to the job manager:

     ```
     wsadmin -profileName myJobManager -lang jython
     ```

3. Review additional connection options for the wsadmin tool.

   You can start the wsadmin scripting client in several different ways. To specify the method for running scripts, perform one of the following wsadmin tool options:

   Run scripting commands interactively

   Run wsadmin with an option other than -f or -c or without an option. The wsadmin tool starts and displays an interactive shell with a wsadmin prompt. From the wsadmin prompt, enter any Jacl or Jython command. You can also invoke commands using the AdminControl, AdminApp, AdminConfig, AdminTask, or Help wsadmin objects. To leave an interactive scripting session, use the quit or exit commands. These commands do not take any arguments.

   The following examples launch the wsadmin tool:

   - Launch the wsadmin tool using Jython:

```
wsadmin -lang jython
```

- Launch the wsadmin tool using Jython when security is enabled:

```
wsadmin -lang jython -user wsadmin -password wsadmin
```

- Launch the wsadmin tool using Jacl with no options:

```
wsadmin -lang jacl
```

Run scripting commands as individual commands

Run the wsadmin tool with the -c option.

The following examples run commands individually:

- Run the list command for the AdminApp object using Jython:

```
wsadmin -lang jython -c "AdminApp.list()"
```

- Run the list command for the AdminApp object using Jacl:

```
wsadmin -c "$AdminApp list"
```

Run scripting commands in a script

Run the wsadmin tool with the `-f` option, and place the commands that you want to run into the file.

The following examples run scripts:

- Run the `a1.py` script using Jython:

```
wsadmin -lang jython -f  a1.py
```

where the `a1.py` file contains the following commands:

```
apps = AdminApp.list()
print apps
```

Run scripting commands in a profile script

A *profile script* is a script that runs before the main script, or before entering interactive mode. You can use profile scripts to set up a scripting environment that is customized for the user or the installation.

By default, the following profile script files might be configured for the `com.ibm.ws.scripting.profiles` profiles property in the *app_server_root*`/properties/wsadmin.properties` file:

- *app_server_root*`/bin/securityProcs.jacl`
- *app_server_root*`/bin/LTPA_LDAPSecurityProcs.jacl`

By default, these files are in ASCII. If you use the `profile.encoding` option to run EBCDIC encoded profile script files, change the encoding of the files to EBCDIC.

To run scripting commands in a profile script, run the wsadmin tool with the -profile option, and include the commands that you want to run into the profile script.

To customize the script environment, specify one or more profile scripts to run.

Do not use parenthesis in node names when creating profiles.

The following examples run profile scripts:

- Run the `a1prof.py` script using Jython:

```
wsadmin -lang jython -profile a1prof.py
```

where the `a1prof.py` file contains the following commands:

```
apps = AdminApp.list()
print "Applications currently installed:\n " + apps
```

- Run the `a1prof.py` script using Jacl:

```
wsadmin -profile a1prof.jacl
```

where the `a1prof.py` file contains the following commands:

```
set apps [$AdminApp list]
puts "Applications currently installed:\n$apps"
```

Results

The wsadmin returns the following output when it establishes a connection to the server processs:

Jython example output:

```
WASX7209I: Connected to process server1
on node myhost using SOAP connector;
The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>
```

Jacl example output:

```
WASX7209I: Connected to process server1
on node myhost using SOAP connector;
The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>
```

Configuring web services applications using wsadmin scripting

You can use the wsadmin scripting tool to complete the several tasks for a web services application.

Before you begin

Develop a web services application. To learn more, see the implementing web services applications information.

About this task

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages.

To use the wsadmin tool to configure a web services application, publish a Web Services Description Language (WSDL) file, or to configure policy sets:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Follow the steps in one of the following topics, depending on what task you want to complete:
   - Publish WSDL files.
   - Configure web services client deployed WSDL file names.
   - Configure web services client preferred port mappings .
   - Configure web services client port information .
   - Configure the scope of a web services port .

Results

You have configured web services applications with the wsadmin tool.

Enabling WSDM using wsadmin scripting

Use the wsadmin tool with the AdminConfig object to enable Web Services Distributed Management (WSDM) in your environment. WSDM is an OASIS approved standard that supports managing resources through a standardized web service interface.

About this task

The WSDM application is installed as a system application and is disabled by default. In order to use the WSDM functionality, use the script in this topic to enable WSDM.

In a multinode environment, the management code runs across a distributed network of Java virtual machines with a central access point as the deployment manager process for the entire network or cell. Enable WSDM on the deployment manager to manage Java virtual machines within a cell. The WSDM application acts as an administrative client to the managed application server. You can manage the WSDM application from the application server on which it is deployed only.

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Determine the configuration ID of the WSDM application.

   Use the getid option for the AdminConfig object to retrieve the configuration ID and set the value to the `deployment` variable, as the following example demonstrates:

```
deployment = AdminConfig.getid('/Deployment:WebSphereWSDM/')
```

3. Determine the deployed object of the WSDM configuration ID.

   Use the showAttribute option for the AdminConfig object to retrieve the deployedObject attribute and set the value to the `deployedObject` variable, as the following example demonstrates:

```
deployedObject = AdminConfig.showAttribute(deployment, 'deployedObject')
```

4. Determine the target mappings for the WSDM deployed object.

   Use the showAttribute option for the AdminConfig object to retrieve the targetMappings attribute and set the value to the targetMappings variable, as the following example demonstrates:

```
targetMappings = AdminConfig.showAttribute(deployedObject, "targetMappings")
```

5. Enable WSDM.

   Assign each mapping to the `target` variable, then set the enable attribute to `true` in the target mapping, as the following example demonstrates:

```
mappings = targetMappings[1:len(targetMappings)-1].split(" ")
AdminConfig.modify(target, '[[enable true]]')
```

6. Save the configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Example

The following samples provide Jython and Jacl scripts that enable WSDM in your environment:

- Using Jython:

```
deployment = AdminConfig.getid('/Deployment:WebSphereWSDM/')
deployedObject = AdminConfig.showAttribute(deployment, 'deployedObject')
targetMappings = AdminConfig.showAttribute(deployedObject, "targetMappings")
mappings = targetMappings[1:len(targetMappings)-1].split(" ")
for target in mappings:
AdminConfig.modify(target, '[[enable true]]')
AdminConfig.save()
```

- Using Jacl:

```
set deployment [$AdminConfig getid /Deployment:WebSphereWSDM]
set deployedObject [$AdminConfig showAttribute $deployment deployedObject]
set targetMappings [lindex [$AdminConfig showAttribute $deployedObject targetMappings] 0]
$AdminConfig modify $targetMappings {{enable true}}
$AdminConfig save
```

Querying web services using wsadmin scripting

You can use the Jython or Jacl scripting language to query for web services properties with the wsadmin tool. Use the commands in the WebServicesAdmin group to list all web services, service references and attributes, find attributes of a specific web service, determine the web service endpoint, and determine the operation name of a Web service.

Before you begin

The wsadmin administrative scripting program supports two scripting languages, Jacl and Jython. The Jacl syntax is deprecated. This topic includes examples written only in the Jython scripting language.

Before you can complete the procedure for the commands in the WebServicesAdmin group, you must launch the wsadmin tool.

About this task

Use the following commands to query for web services and web service attributes. If you receive a NoItemFoundException error, the specified application, module, service, or endpoint cannot be found. Verify that all parameters are correct.

You can optionally specify the client parameter for any command in the WebServicesAdmin command group. The client parameter indicates whether to return service providers or service clients. Specify `false` to request service providers and `true` to request service clients.

Procedure

- Query the configuration for all installed web services for all enterprise applications.

  Enter the following command. You do not need to specify the application parameter for this command.

```
AdminTask.listWebServices()
```

  This command returns all installed web services. The command also returns the application name, module name, service name, and service type for each web service.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [application
application1] [module webapp1.war] [type JAX-WS] ]'
```

- Query the configuration for all installed web services for a specific enterprise application.

  Enter the following command, and specify the name of the application you want to query:

```
AdminTask.listWebServices('[–application application_name -client false]')
```

  This command returns all installed web services for the *application_name* that you specify. The command also returns the application name, module name, service name, and service type for each web service.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [application
application1] [module webapp1.war] [type JAX-WS] ]'
```

- Query the configuration for the web service name and type for an enterprise application.

  Enter the following command, and specify the application name, module name, and web service name. The **client** parameter is optional.

```
AdminTask.getWebService('[–application application_name
-module module_name -service webservice_name -client false]')
```

  The command returns the web service name and web service type.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [type JAX-WS] ]'
```

- Query the configuration for the web service endpoints for an enterprise application.

The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. Enter the following command, and specify the application name, module name, and web service name. The **client** parameter is optional.

```
AdminTask.listWebServiceEndpoints('[–application application_name
 -module module_name -service webservice_name -client false]')
```

This command returns the port on which the web service is installed.

Sample output:

```
'[logicalEndpoint QuotePort01]'
```

- Query the configuration for the web service operation names for an enterprise application.

  Enter the following command, and specify the application name, module name, web service name, and endpoint name. The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. The **client** parameter is optional.

```
AdminTask.listWebServiceOperations('[–application application_name -module module_name
-service webservice_name -logicalEndpoint endpoint_name -client false]')
```

  This command returns all web service operations.

  Sample output:

```
'[operation ivt_app_op1] [operation ivt_app_op2]'
```

- Query the configuration for the service providers, endpoints, and operations from each deployed asset.

  The listServices command provides generic query functions. Use the following command to display information about service providers, endpoints, and operations for enterprise applications and Web Services Notification (WSN) clients. Each parameter is optional. If you do not specify the queryProps parameter, the command returns each service provider in your configuration. If you do not specify the expandResources parameter, the command does not return the logical endpoints or operations for each service.

  The following command example returns each service provider and the corresponding endpoints for the `myApplication` application:

```
AdminTask.listServices('-queryProps "[[CompositionUnit=myApplication][client=false]" -expandResources endpoint']
```

  The following command example returns each JAX-WS service provider and the corresponding endpoints and operations within a cell:

```
AdminTask.listServices('[-queryProps "[[serviceType JAX-WS][client false]]"]')
```

  This command returns the service providers that match the search query.

  Sample output:

```
'[ [service {http://www.ibm.com}EchoService] [assetType [J2EE Application]] [client false]
 [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[ [service {http://www.ibm.com}EchoService12] [assetType [J2EE Application]] [client false]
 [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[service {http://www.ibm.com}PingService] [assetType [J2EE Application]] [client false]
[application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[ [service {http://www.ibm.com}PingService12] [assetType [J2EE Application]] [client false]
[application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS]]'
```

  Note: Use the listServices command to learn about service references. You can use the `serviceRef` property with the queryProps parameter with the listServices command to query all service references or a specific service reference. This parameter is only applicable for service clients. If you specify an asterisk (*) as a wildcard as the name of the service reference, all of the service references for the matching service client are returned. You can also query a specific service reference name by specifying the name of the service reference that you want. To return detailed service reference information for endpoints and operations, specify the expandResource property.

  The following command example lists all service clients and service references in a cell:

```
AdminTask.listServices('[-queryProps [[client true] [serviceRef *]]]')
```

This command returns the service references that match the search query.

In the following sample output, the EchoService and Echoservice12 service clients are listed and these clients do not have service references in the cell. However, TestService and TestService2 service clients do have service references. There is one properties object for each service reference in the service client. The component name is only available for an EJB module.

```
'[ [service {http://www.ibm.com}EchoService] [assetType [J2EE Application]] [client true]
[application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [service {http://www.ibm.com}EchoService12] [assetType [J2EE Application]] [client true]
[application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [service {http://www.ibm.com}TestService] [assetType [J2EE Application]] [client true]
[application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [serviceRef testRef] [assetType [J2EE Application]] [service {http://www.ibm.com}TestService]
[client true] [application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [service {http://www.ibm.com}TestService2] [assetType [J2EE Application]] [client true]
 [application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [serviceRef testRef2] [assetType [J2EE Application]] [service {http://www.ibm.com}TestService2]
[client true] [application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]
```

The following command example lists a specific service reference by expanding endpoints and operations:

```
AdminTask.listServices('[-queryProps [[client true] [application JaxWSServicesSamples]
 [module SampleClientSei.war] [serviceRef testRef] -expandResource logicalEndpoint]')
```

This command returns the specified service references for `testRef` and the serviceRef property is included in each properties object; for example:

```
'[ [serviceRef testRef] [service {http://www.ibm.com}TestService] [assetType [J2EE Application]]
[client true] [application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [serviceRef testRef] [module SampleClientSei.war] [serviceType JAX-WS] [client true]
 [service {http://www.ibm.com}TestService] [assetType [J2EE Application]] [logicalEndpoint portA] [application JaxWSServicesSamples] ]

[ [serviceRef testRef] [module SampleClientSei.war] [serviceType JAX-WS] [client true]
 [service {http://www.ibm.com}TestService] [assetType [J2EE Application]]
[logicalEndpoint portB] [application JaxWSServicesSamples] ]'
```

The following command example returns each WSN service client and the corresponding endpoints and operations within a cell:

```
AdminTask.listServices('[-queryProps "[[serviceType [JAX-WS (WSN)]][clie
nt true]]" -expandResource logicalEndpoint]')
```

This command returns the service providers that match the search query.

Sample output:

```
'[ [service {http://www.ibm.com/websphere/wsn/out/remote-publisher}OutboundRemotePublisherService] [assetType [WSN Service]] [client true]
 [bus bus1] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] ]

[ [assetType [WSN Service]] [service {http://www.ibm.com/websphere/wsn/out/remote-publisher}OutboundRemotePublisherService]
 [bus bus1] [client true] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] [logicalEndpoint OutboundRemotePublisherPort] ]

[ [service {http://www.ibm.com/websphere/wsn/out/notification}OutboundNotificationService] [assetType [WSN Service]] [client true]
 [bus bus1] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] ]

[ [assetType [WSN Service]] [service {http://www.ibm.com/websphere/wsn/out/notification}OutboundNotificationService]
 [bus bus1] [client true] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] [logicalEndpoint OutboundNotificationPort] ]'
```

WebServicesAdmin command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. Use the commands in the WebServicesAdmin group to list all web services, service references and attributes, find attributes of a specific web service, determine the web service endpoint, and determine the operation name of a Web service.

Use the commands in the WebServicesAdmin group to query information for installed web services. For more information about the AdminTask object, see the Commands for the AdminTask object information.

The following commands are available for the WebServicesAdmin group of the AdminTask object:

getWebService

The getWebService command retrieves the attributes for a web service. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the web service. (String, required)

Optional parameters

-client
Specifies whether the web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns web service clients. (Boolean, optional)

Return value

Returns a list of attributes, including service name, whether the service is a provider or client, and service type. The service type attribute is only applicable for service providers.

Batch mode example usage

Using Jython string:

```
AdminTask.getWebService('[–application application_name -module module_name -service webservice_name
 -client false]')
```

Sample output:

```
'[[service {http://www.ibm.com}service1][type JAX-WS][client false]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.getWebService ('-interactive')
```

listServices

The listServices command queries the configuration for services, endpoints, and operations. This command provides more generic query functions than the rest of commands in this command group. It is applicable to Java™ applications as well as other assets such as Web Services Notification (WSN) clients.

Target object

None.

Optional parameters

-queryProps
Specifies properties used to locate the service provider or client of interest. For example, if you specify `[[type=JAX-WS][client=true]]`, the command returns each JAX-WS client reference in the enterprise applications. (Properties, optional)

The -queryProps parameter accepts several properties. You can use one or multiple properties to specify your query criteria. Do not mix the query properties between different asset types. For example, if you specify `application` and `bus`, the command reports an error.

- Specify the following properties with the -queryProps parameter to query enterprise applications:

Table 81. Properties for the -queryProps parameter for enterprise applications. Use these properties to query enterprise applications.

Property and value	Description
`assetType=J2EE Application`	Queries each Java EE application.
`application=`*`application_name`*	Queries a specific Java EE application.
`module=`*`module_name`*	Queries a specific Java EE application module. You must specify the `application` and `module` properties to query for application modules.

- Specify the following properties with the -queryProps parameter to query for WSN clients:

Table 82. Properties to query WSN clients. Use these properties to query WSN clients.

Property and value	Description
`assetType=WSN Service`	Queries each WSN service client.
`bus=`*`bus_name`*	Queries a specific bus.
`WSNService=`*`WSN_service_name`*	Queries a specific WSN service. You must specify the `bus` and `WSNService` properties to query for a specific WSN service.

- Specify the following properties with the -queryProps parameter to query all assets:

Table 83. Properties for the -queryProps parameter to query all assets. Use these properties to query all assets.

Property and value	Description
`serviceType=`*`service_type`*	Queries by service type. Specify `JAX-WS` to query for Java™ API for XML-Based Web Services assets. Specify `JAX-WS (WSN)` to query for Web Services Notification assets.
`client=`*`Boolean`*	Queries for clients or providers. Specify `true` to query for clients. Specify `false` to query for providers.
`service=`*`service_name`*	Queries for the logical endpoints and operations for a specific service.

- Specify the following properties with the -queryProps parameter to query all service references or a specific service reference when `client=true`:

Table 84. Properties for the -queryProps parameter to query service references. Use these properties to query all service references for service clients.

Property and value	Description
serviceRef=*serviceref_name*	Queries for a specific service reference. You can also use an asterisk (*) as a wildcard character to specify to return all the service references under the matching service client.

-expandResource

Specifies whether to return service names only or services and detailed resource information. Specify `logicalEndpoint` or `operation`. If you specify the `logicalEndpoint` value, the command returns the matching services and each endpoint in the services. If you specify the `operation` value, the command returns the matching services and the corresponding endpoints and operations. (String, optional)

You can use the expandResource parameter to return detailed resource information for the endpoints and operations under the service reference.

Return value

The command returns a list of properties for each service, as well as detailed endpoint and operation information if you query for endpoints and operations.

Batch mode example usage

The following examples query each service provider in the `myApplication` application. The examples return the logical endpoints for each service because the -expandResources parameter is set as `logicalEndpoint`, as the following example output demonstrates:

```
[ [service {http://www.ibm.com}EchoService] [assetType [J2EE Application]]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS] ]
[ [assetType [J2EE Application]] [service {http://www.ibm.com}EchoService]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS]
[logicalEndpoint EchoServicePort] ]

[ [service {http://www.ibm.com}PingService] [assetType [J2EE Application]]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS] ]
[ [assetType [J2EE Application]] [service {http://www.ibm.com}PingService]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS]
 [logicalEndpoint PingServicePort] ]
```

Using Jython string:

```
AdminTask.listServices('[-queryProps [[application MyWSApplication][client false]] -expandResource
 logicalEndpoint]')
```

Using Jython list:

```
AdminTask.listServices(['–queryProps', '[[application myApplication][client false]]', '-expandResource',
 'logicalEndpoint'])
```

The following examples query each service client under the `myBus` bus. The examples do not return logical endpoints or operations for each service client because it does not specify the -expandResource parameter.

Using Jython string:

```
AdminTask.listServices('[–queryProps [[bus myBus][client true]] ]')
```

Using Jython list:

```
AdminTask.listServices(['–queryProps', '[[bus myBus][client true]]'])
```

The following examples query service references with the name `testRef` for each service client for the JaxWSServicesSamples application. The examples return detailed resource information for the logical endpoints or operations for each service reference because the -expandResource parameter is specified.

Using Jython string:

```
AdminTask.listServices('[-queryProps [[client true] [application JaxWSServicesSamples] [module SampleClientSei.war]
 [serviceRef testRef] -expandResource logicalEndpoint]')
```

Using Jython list:

```
AdminTask.listServices(['-queryProps', '[[application JaxWSServicesSamples][client true][module SampleClientSei.war]
[serviceRef testRef]', '-expandResource', 'logicalEndpoint'])
```

Sample output:

```
'[ [serviceRef testRef] [service {http://www.ibm.com}TestService] [assetType [J2EE Application]] [client true]
 [application JaxWSServicesSamples] [module SampleClientSei.war] [serviceType JAX-WS] ]

[ [serviceRef testRef] [module SampleClientSei.war] [serviceType JAX-WS] [client true]
 [service {http://www.ibm.com}TestService] [assetType [J2EE Application]] [logicalEndpoint portA]
[application JaxWSServicesSamples] ]

[ [serviceRef testRef] [module SampleClientSei.war] [serviceType JAX-WS] [client true]
 [service {http://www.ibm.com/}TestService] [assetType [J2EE Application]] [logicalEndpoint portB]
 [application JaxWSServicesSamples] ]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listServices('-interactive')
```

listWebServices

The listWebServices command retrieves a list of available web services for one or all applications. If an application name is not supplied, the command lists all of the web services. This command applies to enterprise applications only.

Target object

None.

Required parameters

None.

Optional parameters

-application
Specifies the name of the deployed enterprise application. If you do not specify this parameter, the command returns all web services in the cell. (String, optional)

-client
Specifies whether the web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns web service clients. (Boolean, optional)

Return value

All web services for the application specified. For each web service, the command returns the following attributes and corresponding values: application name, module name, service name, whether the web service is a service provider or client, and service type. The service type is only specified if the web service is a service provider.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServices('[–application application1 -client false]')
```

Using Jython list:

```
AdminTask.listWebServices(['–application', 'application1', '-client', 'false'])
```

Sample output:

```
'[[service {http://www.ibm.com}service1][application application1][module webapp1.war][type JAX-WS][client false]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServices('-interactive')
```

listWebServiceEndpoints

The listWebServiceEndpoints command returns a list of logical endpoints for a web service. The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the web service. (String, required)

Optional parameters

-client
Specifies whether the web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns web service clients. (Boolean, optional)

Return value

Returns the logical endpoint name for the web service specified.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServiceEndpoints('[–application application_name -module module_name
-service webservice_name -client false]')
```

Using Jython list:

```
AdminTask.listWebServiceEndpoints(['–application', 'application_name', '-module', 'module_name',
 '-service', 'webservice_name', '-client', 'false'])
```

Sample output:

```
'[[logicalEndpoint QuotePort01]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServiceEndpoints('-interactive')
```

listWebServiceOperations

The listWebServiceOperations command returns a list of web service operations. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the web service. (String, required)

-logicalEndpoint
The port name in the Web Services Description Language (WSDL) document. (String, required)

Optional parameters

-client
Whether the web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns web service clients. (Boolean, optional)

Return value

Returns the operation name for the web service specified.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServiceOperations('[–application application_name -module
module_name -service webservice_name -client false
 -logicalEndpoint endpoint_name]')
```

Using Jython list:

```
AdminTask.listWebServiceOperations(['–application', 'application_name', '-module',
'module_name', '-service', 'webservice_name', '-client',
 'false', '-logicalEndpoint', 'endpoint_name'])
```

Sample output:

```
'[[operation ivt_app_op1][operation ivt_app_op2]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServiceOperations('-interactive')
```

Configuring a web service client deployed WSDL file name using wsadmin scripting

When a web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the web module or Enterprise JavaBeans (EJB) module, including the client bindings.

Before you begin

You should have the web service application ready for deployment or already deployed the web service into WebSphere Application Server before starting this task.

To complete this task, you need to know the topology of the URL endpoint address of the web services servers and which web service the client depends upon. You can view the deployment descriptors in the administrative console to find topology information. To learn more, see the viewing web services server deployment descriptors in the administrative console information.

For more information about the wsadmin tool options, see the information on options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands.

About this task

The client bindings define the Web Services Description Language (WSDL) file name and preferred ports. The relative path of a web service in a module is specified within a compatible WSDL file that contains the actual URL to be used for requests. The address is only needed if the original WSDL file did not contain a URL, or when a different address is needed. For a service endpoint with multiple ports, you need to define an alternative WSDL file name.

The following steps describe how to edit bindings for a web service after these bindings are deployed on a server. When one web service communicates with another web service, you must configure the client bindings to access the downstream web service.

You can use the **WebServicesClientBindDeployedWsdl** command-line option in this task to change the endpoint. One of the benefits to using the command-line option is that you can avoid uninstalling, modifying enterprise archive (EAR) files and reinstalling applications to make binding configuration changes. Another benefit is the ability to customize the web service bindings applications for different environments during installation, and to avoid the need to create different application EAR files for each version.

Several versions of WSDL files, each with different service endpoints, can be provided during the development and assembly of a web service module that is acting as a client to a web service. During or after the installation, when you are configuring the installed application, the WebServicesClientBindDeployedWsdl option can be used to specify which of the WSDL files to use.

Because the WSDL file defines all the service endpoints or implementations for all of the port types and ports that the client can use, the deployed WSDL file can group a set of choices into one WSDL. You can override the endpoint by port.

You can use Jacl or Jython scripts, but this task assumes that you are using Jacl. For more information about deploying and managing using scripting, see the getting started with scripting information.

To configure client bindings with the wsadmin tool, proceed with the following steps:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.

2. At the **wsadmin** command prompt, enter the command syntax. You can use install, installInteractive, edit or editInteractive options. The following example presents the syntax:

   ```
   $AdminApp edit <app_name> {
   -WebServicesClientBindDeployedWsdl {{<module_name> <EJB_name> <web_service>
   <deployed_WSDL_filename>}...}
   ```

   The example shows multiple nodules and URL endpoints because you can edit multiple URL fragments. where:
   - *app_name* is the application name, for example `WebServicesSample.ear`
   - *module_name* is the EJB or web module name, for example `AddressBookW2JE.jar`
   - *EJB_name* is the name of the EJB if the module is not a web module, for example Exchange
   - *web_service* is the name of the web service, for example service/StockQuoteService
   - *deployed_WSDL_filename* identifies the WSDL file, relative to this module, for example, `META-INF/wsdl/AlternativeStockQuoteFetcher.wsdl`
3. Save the configuration changes with the **$AdminConfig save** command:

Results

Your web service client bindings are configured.

Example

The following example presents the application, module and deployed WSDL file name as it is written in the command line:

```
$AdminApp edit MultiEjbJar {-WebServicesClientBindDeployedWSDL {{ejbclientonly.jar Exchange
service/StockQuoteService META-INF/wsdl/AlternateStockQuoteFetcher.wsdl}...}}
```

What to do next

Now you can finish any other configurations, start or restart the application, and verify the expected behavior of the web service.

Configuring web service client-preferred port mappings using wsadmin scripting

A client port type can be configured with ports that have different qualities of service. You can use the **WebServicesClientBindPreferredPort** command-line option to specify which port you want to use.

Before you begin

If you have not deployed the enterprise archive (EAR) file yet, you need to have it ready or already deployed to the application server.

About this task

For each port type that is configured, one or more ports are available that implement that port type. When a web service client calls a getPort method, the preferred port mapping determines which port to use. This determination occurs when more than one port can satisfy the getPort method call, such as, a getPort call that specifies the port type, but not the port. For example, suppose the web service client is configured to use both Java Message Service (JMS) and an HTTP implementation. During installation or management, you can use the **WebServicesClientBindPreferredPort** command to configure the preferred port of the application to use the transport of choice.

To configure the preferred port mapping with the wsadmin tool proceed with the following steps:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Configure web service client-preferred port mappings.

   To use the existing listener port instead of using or creating a new activation specification, determine whether the EJB JAR version is lower than 2.1. The system automatically creates and uses an activation specification when you specify the -usedefaultbindings option to deploy an application. If an activation specification exists, the system ignores the listener port, and instead uses the activation specification. To deploy an application with an EJB JAR version greater than or equal to 2.1 using the defined listener ports instead of a new activation specification, set the com.ibm.websphere.management.application.dfltbndng.mdb.preferexisting system property to `true` in the `wsadmin.properties` file in the properties directory of the profile of interest.

   Use the install, installInteractive, edit or editInteractive options to configure the web service client-preferred port mappings, as the following syntax demonstrates:

   - Using Jython:

     ```
     AdminApp.install('app_name', '[-usedefaultbindings -deployejb
     -WebServicesClientBindPreferredPort {{module_name EJB_name Web_service port_type
     port_name]')
     ```

   - Using Jacl:

     ```
     $AdminApp install app_name {-usedefaultbindings -deployejb
     -WebServicesClientBindPreferredPort {{module_name EJB_name Web_service port_type
     port_name}}
     ```

   The example shows multiple modules and URL endpoints, because you can edit multiple URL fragments where:

   - *app_name* is the application name, for example `MultiEjbJar.ear`
   - *EJB_name* is the name of the enterprise bean module that is not a web module, for example, Exchange
   - *module_name* is the module name, for example `ejbclientonly.jar`
   - *Web_service* is the name of the web service, for example `service/StockQuoteService`
   - *port_type* is the port type information, for example `{http://stock.multiejbjar.test.wsfvt.ws.ibm.com}StockQuote`
   - *port_name* is the port name, for example `{http://stock.multiejbjar.test.wsfvt.ws.ibm.com}StockQuote`

Results

You have configured web service client-preferred port mappings with the wsadmin tool.

Example

The following example includes the application, module, Web service, port type and port information as it is written in the command line:

```
$AdminApp install MultiEjbJar.ear {-WebServicesClientBindPreferredPort {{ejbclientonly.jar
Exchange service/StockQuoteService {http://stock.multiejbjar.test.wsfvt.ws.ibm.com}StockQuote
{http://stock.multiejbjar.test.wsfvt.ws.ibm.com}StockQuote}...}}
```

The port type information that drives the creation of the WebServicesClientBindPreferredPort option data resides in the client WSDL file. Because valid preferred port mappings are restricted to ports that implement the interface of the port type, validation requires the implementation type of each port. The client WSDL file must be accessed to determine both the type and the implementation information.

The client WSDL file name is in the ServiceRef attribute of the web service client deployment descriptor. Depending on the module type and version, the client deployment descriptor is located in either the `application-client.xml` file; the `web.xml` file, or the `ejb-jar.xml` file. If you are using J2EE 1.3, the client

deployment descriptor information is located in the `webservices.xml` file.

What to do next

Now you can finish any other configurations, start or restart the application, and verify expected behavior of the web service.

Configuring web service client port information using wsadmin scripting

A web service can have multiple ports. You can view and configure the port attributes for each defined web service port.

Before you begin

If you have not deployed the enterprise archive (EAR) file yet, you need to have it ready or already deployed to the application server.

About this task

You can configure binding attributes that are associated with the web service client port, including synchronization timeout, overridden endpoint URL and transport attributes with the **WebServicesClientBindPortInfo** command-line option. A typical usage scenario for this command-line option is to customize the timeout value of the client so that the client waits longer when it is configured to use a Java Message Service (JMS) transport to access a web service.

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Configure the web service client port information.

   To use the existing listener port instead of using or creating a new activation specification, determine whether the EJB JAR version is lower than 2.1. The system automatically creates and uses an activation specification when you specify the -usedefaultbindings option to deploy an application. If an activation specification exists, the system ignores the listener port, and instead uses the activation specification. To deploy an application with an EJB JAR version greater than or equal to 2.1 using the defined listener ports instead of a new activation specification, set the com.ibm.websphere.management.application.dfltbndng.mdb.preferexisting system property to `true` in the `wsadmin.properties` file in the properties directory of the profile of interest.

   Use install, installInteractive, edit or editInteractive options to configure the web service client port information, as the following example demonstrates:

   ```
   $AdminApp install app_name {-usedefaultbindings
   -deployejb -WebServicesClientBindPortInfo {{module_name EJB_name
   Web_service port timeout basic_authentication_id basic_authentication_password
   SSL_alias overridden_endpoint overridden_binding_namespace }...}}
   ```

   The previous example indicates that the port information of multiple ports can be changed using one WebServicesClientBindPortInfo command, where:

   - *app_name* is the application name, for example, `MultiEjbJar.ear`
   - *module_name* is the module name, for example `ejbclientonly.jar`
   - *EJB_name* is the name of the EJB that is not a web module, for example, Exchange
   - *Web_service* is the name of the web service, for example service/StockQuoteService
   - *port* is the name of the port, for example StockQuote
   - *timeout* specifies the number of seconds that the client waits for a response
   - *basic_authentication_id* is the basic authentication transport ID
   - *basic_authentication_password* is the basic authentication transport password

- *SSL_alias* identifies the SSL alias for the port
- *overridden_endpoint* is the name of the endpoint that is used to override the current endpoint
- *overridden_binding_namespace* specifies the WSDL file binding namespace URI to use with the port

3. Save the configuration changes with the **$AdminConfig save** command:

Results

The client port information that is associated with the web service client port are configured.

Example

```
$AdminApp installInteractive MultiEjbJar.ear {-WebServicesClientBindPortInfo
 {{ejbclientonly.jar Exchange service/StockQuoteService StockQuote 6000 jsmith js9password level3ssl
 http://fastball.houston.ibm.com/newURL http://fastball.houston.ibm.com/newBindName}}
 {ejbclientonly.jar Exchange service/StockQuoteService StockQuote2 9000 {}{}{}{}{}}}
```

What to do next

Now you can finish any other configurations, start or restart the application, and verify expected behavior of the web service.

Configuring the scope of a web service port using wsadmin scripting

When a web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the web module or Enterprise JavaBeans (EJB) module, including implementation scope, client bindings and deployment descriptor information. There are three levels of scope that can be set: application, session and request.

Before you begin

If you have not deployed the enterprise archive (EAR) file yet, you need to have it ready or already deployed to the application server.

About this task

The primary purpose of this task is to enable the configuration of the web service port scope. The scope originally specified when the JavaBeans object is enabled as a web service during the development process can be changed with the **WebServicesServerBindPort** command. The scope attribute does not apply to web services that use Java Message Service (JMS) transport or to enterprise beans.

Web Services for Java 2 platform Enterprise Edition (J2EE) specifies that web services implementations must be stateless. Therefore, to maintain specification compliance, the scope can remain at the application level because the state relevant to the individual sessions level or the requests level is not supposed to be maintained in the implementation. If you want to deviate from the specification and want to access a different JavaBeans instance, because you are looking for information that is located in another JavaBeans, the scope settings need to change.

The setting that you configure for the scope determines how frequently a new instance of a service implementation class is created for the web service ports in a module. The application scope causes the same instance of the implementation to be used for all requests on the application. The session scope causes the same instance to be used for all requests in each session. The request scope causes a new instance to be used for every request. For example, with the scope set to application, every message that comes to the server accesses the same Java bean instance.

To change the scope setting through the wsadmin tool:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Configure the scope of the web service port.

   To use the existing listener port instead of using or creating a new activation specification, determine whether the EJB JAR version is lower than 2.1. The system automatically creates and uses an activation specification when you specify the -usedefaultbindings option to deploy an application. If an activation specification exists, the system ignores the listener port, and instead uses the activation specification. To deploy an application with an EJB JAR version greater than or equal to 2.1 using the defined listener ports instead of a new activation specification, set the com.ibm.websphere.management.application.dfltbndng.mdb.preferexisting system property to `true` in the `wsadmin.properties` file in the properties directory of the profile of interest.

   Use the install, installInteractive, edit, or editInteractive options to configure the scope of the web service port, as the following syntax demonstrates:

   ```
   $AdminApp install app_name {-usedefaultbindings -deployejb
   -WebServicesServerBindPort {{<module_name> <Web_service> <port><scope_setting>}...}
   ```

   The previous example indicates that the scope of multiple ports can be changed using one WebServicesServerBindPort command, where:
   - *app_name* is the application name, for example `WebServicesSample.ear`
   - *module_name* is the module name, for example `AddressBookW2JE.jar`
   - *Web_service* is the name of the web service, for example `AddressBookW2JE service/ WSLoggerService2`
   - *port* is the name of the port, for example `AddressBook`
   - *scope_setting* is the level of setting for scope, for example `Session`

Results

The scope for a web service port is configured.

Example

The following example of the presents the application, module, web service, port and scope as it is written in the command line:

```
$AdminApp install WebServicesSamples.ear {-usedefaultbindings -deployejb -deployws
-WebServicesServerBindPort {{AddressBookJ2WB.war AddressBookService AddressBook request}
{AddressBookW2JB.war AddressBookService AddressBook application}}}
```

What to do next

Now you can finish any other configurations, start or restart the application, and verify expected behavior of the web service.

Publishing WSDL files using wsadmin scripting

The Web Services Description Language (WSDL) files in each web services-enabled module are published to the file system location you specify. You can provide these WSDL files in the development and configuration process of web services clients so that they can invoke your web services.

Before you begin

Before you publish a WSDL file, you can configure web services to specify endpoint information in the form of URL fragments to enable full URL specification of WSDL ports. Refer to the tasks describing configuring endpoint URL information.

To publish a Web Services Description Language (WSDL) file you need an enterprise application, also known as an enterprise archive (EAR) file, that contains a Web services-enabled module and has been

deployed into WebSphere Application Server. To learn more, see the deploying web services applications onto application servers information.

About this task

The purpose of publishing the WSDL file is to provide clients with a description of the web service, including the URL identifying the location of the service.

After installing a web services application, and optionally modifying the endpoint information, you might need WSDL files containing the updated endpoint information. You can obtain the updated WSDL files by publishing them to the file system. If you are a client developer or a system administrator, you can use WSDL files to enable clients to connect to a web service.

The wsadmin tool can publish the WSDL files in either local, for example, -conntype NONE, or remote mode. However, in local mode, locate the target application at the same node where the wsadmin command is invoked.

The following steps assume that the application has been deployed and that the application server is running.

Procedure

1. Start the wsadmin tool from the command prompt using the following commands:
   - *profile_root*/bin/wsadmin
2. At the wsadmin command prompt, enter one of the two commands:
   - `$AdminApp publishWSDL`*app_name path_name*
   - `$AdminApp publishWSDL`*app_name path_name soapAddressPrefixes*

   where:
   - *app_name* is the application name
   - *path_name* is the absolute path to the compressed file that contains the published WSDL files. The compressed file is saved on the machine that runs WebSphere Application Server, therefore, if the server is running on a different machine, you need to obtain the compressed file from that machine. The directory structure of the resulting compressed file is based on the following information:

     ```
     Application_file_name/module_file_name/META-INF/ or WEB-INF/wsdl/WSDL_file_name
     ```

     See the usage scenario for an example of this directory structure.
   - *soapAddressPrefixes* is a parameter of the form {{*module* {{*binding partial-url*}}}}. This parameter describes the partial URL information for each binding on a per-module basis for the application.
     - *module* identifies the name of the module
     - *binding* is either http or jms (both are in lower case)
     - *partial-url* is the partial SOAP address for the associated SOAP binding. For an HTTP binding, the form is `http://host:port/` or `https://host:port`.

       For Java Message Service (JMS) bindings, the form is

       ```
       jms:/queue?destination=dest&connectionFactory=cf
       ```

       or

       ```
       jms:/topic?destination=dest&connectionFactory=cf
       ```

   Use the **$AdminApp publishWSDL *app_Name path_Name*** command to publish WSDL files with default endpoint URL addresses. If you want to modify the SOAP address prefixes of the WSDL file, use the other form of the command.

   Use the **$AdminApp publishWSDL *app_Name path_Name* {{*module* {{*binding partial-url*}}}}** command to customize the WSDL SOAP address for each module. You can specify a different address prefix for each SOAP binding.

Results

The WSDL files from web services are published to a specified compressed file. The compressed file can be used to create a web service client that accesses the deployed service. The published WSDL files do not contain Enterprise JavaBeans (EJB) binding information.

Example

The command to publish WSDL files for a web service named WebServicesSamples can be $AdminApp publishWSDL WebServicesSamples c:/temp/samplesWsdl.zip

or

$AdminApp publishWSDL WebServicesSamples c:/temp/sampleswsdl.zip { {AddressBookJ2WB.war {{http http://localhost:9080}}} {StockQuote.jar {{http https://localhost:9443}}} }

The directory structure for this created compressed file is

```
WebServicesSamples.ear/StockQuote.jar/META-INF/wsdl/StockQuoteFetcher.wsdl
WebServicesSamples.ear/AddressBookW2JE.jar/META-INF/wsdl/AddressBookW2JE.wsdl
WebServicesSamples.ear/AddressBookJ2WE.jar/META-INF/wsdl/AddressBookJ2WE.wsdl
WebServicesSamples.ear/AddressBookJ2WB.war/WEB-INF/wsdl/AddressBookJ2WB.wsdl
WebServicesSamples.ear/AddressBookW2JB.war/WEB-INF/wsdl/AddressBookW2JB.wsdl
```

What to do next

Develop a web services client or configure the endpoint information for an existing web service.

Configuring application and system policy sets for web services using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

Before you begin

Develop a web services application. For additional information, see the web services applications topics in the information center.

If you develop an application that uses a custom policy set, the custom policy set configuration is not included in the application enterprise archive (EAR) file. Install the application and import the custom policy set separately.

About this task

The commands in the PolicySetManagement group for the AdminTask object configure both application and system policy sets. Use the following tasks to configure and manage policy sets for your web services.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured

general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 and trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 and later bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 and later environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

Procedure

- Use the PolicySetManagement group of commands to configure application and client policy sets:
  - Create a new policy set or copy an existing policy set.
  - Add policies to your policy set.
  - Attach your policy set to an application, web service, endpoint, or operation.

  - Customize cell-wide, server-specific, or application binding configurations.
  - Manage and edit your policy set configurations.
    - Edit, enable, disable, or remove policies.
    - Add, edit, or remove policy set attachments.
    - Export and import policy sets.
    - Delete policy sets.
- Use the PolicySetManagement group of commands to configure system policy sets.
  - Create a new system policy set or copy an existing system policy set.
  - Add policy types for your policy set.
  - Add trust service attachments.
  - Customize binding configurations.
  - Manage and edit your policy set configurations.
    - Edit, enable, disable or remove policies.
    - Add, edit, or remove policy set attachments.
    - Export and import policy sets.
    - Delete policy sets.

Creating policy sets using wsadmin scripting

Create policy sets to centrally manage policies that are customized for your web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

Before you begin

In order to complete this task, you must use the Administrator role with cell-wide access when administrative security is enabled.

About this task

There are three ways to create a new policy set using the wsadmin tool. You can create a new policy set and its configuration, copy an existing policy set, or import a policy set.

When you create a new policy set, you must add policies. If you copy an existing policy set, you can transfer the policies and attachments that are configured on the existing policy set. The command examples in this topic use batch mode syntax. You can use the -interactive option with all commands in the PolicySetManagement group.

Procedure

- Create a new policy set using the Jython scripting language.
  1. Start the wsadmin scripting tool.
  2. Determine the policy requirements for your web services.
  3. Enter the command syntax to create a new policy set with a given name.

     Based on your configuration, there are two types of policy sets to create. You can use both application and system policy sets with Java API for XML-Based Web Services (JAX-WS) applications. Use the -policySetType parameter to specify the type of policy set. To create an application policy set, specify `application` for the value of the -policySetType parameter. To create a policy set for the trust service, specify `system` or `system/trust` for the -policySetType parameter. For WS-MetadataExchange attachments, specify `system` for the -policySetType parameter. The -policySetType parameter is optional. The wsadmin tool creates an application policy set if the -policySetType parameter is not specified.

Enter the following command to create an application policy set:

```
AdminTask.createPolicySet('[-policySet PolicySet1 -description policySet_description]')
```

Enter the following command to create a policy set for the trust service:

```
AdminTask.createPolicySet('[-policySet PolicySet1 -description policySet_description -policySetType system]')
```

The command returns a success or failure message.

4. Add policies for your new policy set. Use this step to add a policy with default values for the specified policy set.

   Enter the following command to add and enable a policy:

```
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType policyType_name]')
```

   Enter the following command to add and disable a policy. Your configuration changes are contained within the policy set, but will have no effect on the system if the -enabled parameter is set to `false`.

```
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType policyType_name -enabled false]')
```

   The command returns a success or failure message. Repeat this step to create additional policies for your configuration.

5. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

- Copy an existing policy set using the Jython scripting language.
  1. Start the wsadmin scripting tool.
  2. Determine the policy requirements for your web services.
  3. Enter the command syntax to copy an existing policy set:

     Set the -transferAttachments parameter to `true` to transfer the attachments from the existing policy set to the new policy set. The default value for the -transferAttachments parameter is `false`.

     Enter the following command to create the new policy set and to transfer the attachments of the existing policy set:

```
AdminTask.copyPolicySet('[-sourcePolicySet existingPolicySet_name -newPolicySet PolicySet1
-newDescription PolicySet1_description -transferAttachments true]')
```

     The command returns a success or failure message.

  4. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

- Import a policy set from an archive file or import a default policy set using the Jython scripting language.
  1. Start the wsadmin scripting tool.
  2. Determine the policy requirements for your web services.
  3. Import a policy set.

     Use the importPolicySet command to import the archive file containing the policy set configuration of interest to the destination environment. Specify the verifyPolicySetType parameter to verify that the policy set to import matches a specific type. Set the value as `application`, `system`, or `system/trust` to specify the policy set type. You cannot import a policy set onto a server or client environment if the policy set already exists in the destination environment.

     For example, the following command creates a customSecureConversation policy set from the customSC.zip archive file:

```
AdminTask.importPolicySet('[-importFile /IBM/WebSphere/AppServer/bin/customSC.zip -verifyPolicySetType
 system/trust]')
```

     Additionally, you can also use the importPolicySet command to import a default policy set onto a server environment, as the following example demonstrates:

```
AdminTask.importPolicySet('[-defaultPolicySet SecureConversation -policySet copyOfdefaultSC
 -verifyPolicySetType system]')
```

The command returns a success or failure message.

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

If you receive a success message after entering the commands, you can now manage a policy set that is customized for your web services applications. You can further configure the policy set and policies.

What to do next

Use the **validatePolicySet** command to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Updating policy set attributes using wsadmin scripting

Use policy sets to centrally manage policies that are customized for your web services. Use the Jython or Jacl scripting language with the wsadmin tool to update policy set attributes. You can also query the configuration for an existing policy set and respective attributes.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 85. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to update policy sets.
Configurator	The Configurator role cannot update policy sets.
Deployer	The Deployer role cannot update policy sets.
Operator	The Operator role cannot update policy sets.
Monitor	The Monitor role cannot update policy sets.

About this task

After creating a new policy set, you can update your policy set configuration with the updatePolicySet command. The command uses a properties object to update all attributes or a subset of attributes for the specified policy set.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
2. View the current attribute values for the policy set of interest.

   Enter the following command to display the attribute values of the *policySet1* policy set:

```
AdminTask.getPolicySet('[-policySet policySet1]')
```

3. Modify attributes for the policy set of interest.

   Enter the following command to modify the `type` and `description` attributes for the *policySet1* policy set.

```
AdminTask.updatePolicySet('-policySet policySet1 -attributes "[ [type application] [description
[my custom policy set to manage WSSecurity]] ]"')
```

You can also use the -interactive option to update a policy set. To start -interactive mode, enter this command:

```
AdminTask.updatePolicySet('-interactive')
```

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

If you received a success message after entering the commands, manage and customize the policy set for your web services applications.

What to do next

After updating policy set attributes, you can further configure policies and policy set attachments using the commands and parameters for the PolicySetManagement group of the AdminTask object.

Adding and removing policies using wsadmin scripting

You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Additionally, if administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 86. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create and remove policies.
Configurator	The Configurator role cannot create or remove policies.
Deployer	The Deployer role cannot create or remove policies.
Operator	The Operator role cannot create or remove policies.
Monitor	The Monitor role cannot create or remove policies.

About this task

Policies define which Qualities of Service (QoS) to manage within a policy set. Policy definitions are based on the standards set by the Organization for the Advancement of Structured Information (OASIS) and Web Services Security specifications.

For application policy sets, you can add the following policies:

- WSSecurity
- WSReliableMessaging
- WSAddressing
- HTTPTransport
- SSLTransport

- WSTransaction
- JMSTransport
- CustomProperties

For system policy sets, you can add the following policies:
- WSSecurity
- WSAddressing
- HTTPTransport
- SSLTransport
- WS-MetadataExchange
- JMSTransport
- CustomProperties

Use the following steps to add or remove policy types from your policy set configurations:

Procedure

- Add a policy to a policy set. Use this section to add a policy with default values to the specified policy set. You can create and enable or create and disable the policy.
  1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
  2. List all policies for a specified policy set.

     Enter the following command and specify the policy set of interest to list all policies that have been added to the policy set:

     ```
     AdminTask.listPolicyTypes('[-policySet PolicySet1]')
     ```

     Enter the following command to list all the available policies:

     ```
     AdminTask.listPolicyTypes()
     ```
  3. Add the policy to your configuration.

     Enter the following command to add and enable a policy:

     ```
     AdminTask.addPolicyType('[-policySet PolicySet1
     -policyType policyType_name]')
     ```

     Enter the following command to add and disable a policy. Your configuration changes are contained within the policy set, these changes do not effect the system if the -enabled parameter is set to `false`.

     ```
     AdminTask.addPolicyType('[-policySet PolicySet1
     -policyType policyType_name -enabled false]')
     ```
  4. Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```
  5. For your configuration changes to take effect, restart all applications with attachments to the policy set.

  The command returns a success or failure message. Repeat this step to create additional policies for your configuration.
- Remove a policy from the policy set configuration. The deletePolicyType command removes the specified policy from the policy set. Applications with attachments to the policy set are not affected until the application restarts.
  1. Start the wsadmin scripting tool.
  2. Enter the following command to list all policies for the policy set of interest:

     ```
     AdminTask.listPolicyTypes('[-policySet PolicySet1]')
     ```
  3. Enter the following command to remove the policy:

     ```
     AdminTask.deletePolicyType('[-policySet PolicySet1
     -policyType policyType_name]')
     ```

The command returns a success or failure message.

4. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

5. For your configuration changes to take effect, restart all applications with attachments to the policy set.

What to do next

Use the validatePolicySet command to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Editing policy configurations using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to edit policy configurations for your policy sets.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 87. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to modify policies.
Configurator	The Configurator role cannot modify policies.
Deployer	The Deployer role cannot modify policies.
Operator	The Operator role cannot modify policies.
Monitor	The Monitor role cannot modify policies.

About this task

Policies define the type of policy to manage within a policy set. Policies are based on the Quality of Services (QoS), such as Web Services Security (WS-Security) and Web Services Addressing (WS-Addressing). Policy definitions are based on the standards set by the Organization for the Advancement of Structured Information (OASIS) and WS-Security specifications.

Use the following steps to edit existing policies in your policy set configurations:

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language To learn more, see the starting the wsadmin scripting client information.
2. Determine which policy set to edit.

   To view a list of policies on a policy set, enter the listPolicyTypes command, specifying the policy set of interest.

```
AdminTask.listPolicyTypes('[-policySet PolicySet1]')
```

Enter the listPolicyTypes command without the policySet parameter to view a list of available policies for all policy sets in your configuration:

```
AdminTask.listPolicyTypes()
```

3. Review the policy attributes to edit.

   Enter the getPolicyType command, specifying the policy and associated policy set of interest.

```
AdminTask.getPolicyType('[-policySet PolicySet1 -policyType myPolicyType]')
```

4. Modify the policy attributes.

   Use the setPolicyType command to update the policy configuration. Update one or multiple attributes by passing a properties object for the -attributes parameter. The properties for the -attributes parameter are dependent on the policy type specified in the `-policyType` parameter. The following example modifies the `enabled` and `provides` properties:

```
AdminTask.setPolicyType('[-policySet PolicySet1 -policyType myPolicyType
 -attributes "[[enabled true][provides security]]"')
```

5. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

6. For your configuration changes to take effect, restart all applications with attachments to the policy set.

What to do next

Use the validatePolicySet to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Enabling secure conversation using wsadmin scripting

Use this topic and the commands in the SecureConversation group of the AdminTask object to enable secure conversation client cache by creating a new policy set and bindings to attach to your applications.

Before you begin

Verify that the SecureConversation policy set is available in your configuration. By default, the SecureConversation policy set is not available. Use the importPolicySet command to import the SecureConversation policy to your configuration, as the following example demonstrates:

```
AdminTask.importPolicySet('-defaultPolicySet SecureConversation')
```

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

About this task

This topic uses the default SecureConversation policy set and default WS-Security and TrustServiceSecurityDefault bindings to enable secure conversation.

The default SecureConversation policy set contains an application policy with the symmetric binding, and a bootstrap policy with the asymmetric binding. The application policy secures application messages. The bootstrap policy secures RequestSecurityToken (RST) messages. The trust service, which issues security context token providers, uses the TrustServiceSecurityDefault system policy and the TrustServiceSecurityDefault bindings. The trust policy secures RequestSecurityTokenResponse (RSTR) messages. If you modify the bootstrap policy, you must also modify the trust policy so that both of the configurations match.

Note: Use the following steps in development and test environments only. The WS-Security bindings in this procedure contain sample key files that you must customize before using the bindings in a production environment. Create custom bindings for your production environment.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
2. Copy the existing SecureConversation policy set.

   Use the following command example to create a new policy set by copying the existing SecureConversation policy set:

```
AdminTask.copyPolicySet('[-sourcePolicySet SecureConversation -newPolicySet CopyOfSCPolicySet]')
```

3. Change the binding for the global security domain. If you chose the **Create the server using the development template** option when you created your profile with the Profile Management Tool or the manageprofiles command utility, you can optionally skip this step.
   a. List each WS-Security policy attribute.

      To modify the binding for the global security domain, use the getDefaultBindings command to determine the binding that is set as the default for the provider or client, as the following example demonstrates:

```
AdminTask.getDefaultBinding('-bindingType provider')
```

   b. Display the attributes for the binding.

      Use the getBinding command to display the current attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-bindingLocation "" -bindingName myBinding')
```

   c. Modify the outbound configuration for the protection token.

      Use the following commands to modify the outbound configuration for the protection token:

```
cmd1_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.key.name [CN=Bob,O=IBM, C=US]] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.keystore.storepass storepass] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.keystore.type JCEKS] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler.key.alias bob]
[application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler.keystore.path ${USER_INSTALL_ROOT}
/etc/ws-security/samples/enc-sender.jceks] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "" -attributes cmd1_attributes_value
-attachmentType application]')

cmd2_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore
.path ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks] [application.securityoutboundbindingconfig
.tokengenerator_0.callbackhandler.keystore.storepass client] [application.securityoutboundbindingconfig
.tokengenerator_0.callbackhandler.key.name [CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP]] [application
.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.keypass client] [application
.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.alias soaprequester]
[application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type JKS] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "" -attributes cmd2_attributes_value
-attachmentType application]')
```

4. Optional: Modify the TrustDefaultBindings binding. If you chose the **Create the server using the development template** option when you created your profile with the Profile Management Tool or the manageprofiles command utility, you can optionally skip this step.

   If the TrustDefaultBindings are not yet customized, use the following commands to modify the TrustDefaultBindings binding:

```
cmd3_attributes_value = "[  [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore
.storepass storepass] [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.key.alias bob]
[application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore.type JCEKS] [application
.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore.path ${USER_INSTALL_ROOT}/etc
/ws-security/samples/enc-sender.jceks] [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler
.key.name [CN=Bob, O=IBM, C=US]] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "[attachmentId 2]"
-attributes cmd3_attributes_value -attachmentType system/trust]')

cmd4_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.path
```

```
${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks] [application.securityoutboundbindingconfig.tokengenerator_0
.callbackhandler.keystore.storepass client] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler
.key.name [CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP]] [application.securityoutboundbindingconfig.tokengenerator_0
.callbackhandler.key.keypass client] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key
.alias soaprequester] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type JKS] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "[attachmentId 2]"
-attributes cmd4_attributes_value -attachmentType system/trust]')
```

5. Attach the policy set and binding to the application.

   Use the attachmentType parameter for the createPolicySetAttachment command to specify if your application is a service client or a service provider. Use the following commands to attach the *CopyOfSCPolicySet* policy set to the *myTestApp* service client application:

```
AdminTask.createPolicySetAttachment('[-applicationName myTestApp -policySet CopyOfSCPolicySet
-resources WebService:/ -attachmentType client]')
```

   Use the following commands to attach the *CopyOfSCPolicySet* policy set to the *myTestApp* service provider application:

```
AdminTask.createPolicySetAttachment('[-applicationName myTestApp -policySet CopyOfSCPolicySet
-resources WebService:/ -attachmentType application]')
```

   This step automatically assigns the bindings.

Results

Your secure conversation configuration is updated in the WSSCCache.xml file located in the cell level directory.

What to do next

Manage your secure conversation configurations with the SecureConversation command group for the AdminTask object.

Managing WS-Security distributed cache configurations using wsadmin scripting

The distributed cache stores tokens on the client. Use this topic and the commands in the WSSCacheManagement group of the AdminTask object to query, update, and remove custom and non-custom properties for the distributed cache configuration.

Before you begin

Configure a policy set with WS-Security enabled.

About this task

The distributed cache stores tokens on both distributed and local clients. WebSphere Application Server supports only the security context token for the WS-Trust security token service client and the security trust service components.

You can use the administrative console or the wsadmin tool to manage your secure conversation distributed cache configuration. You can use the wsadmin tool and the Jython scripting language syntax to:

- Query your current distributed cache configuration settings.
- Set the value for the renewal time after token expiration.
- Enable or disable distributed cache for clustered servers.
- Add custom properties to your configuration.
- Remove custom properties from your configuration.

Procedure

- Query the configuration for your existing distributed cache configuration.

You can retrieve a list of your current distributed cache configuration settings and custom properties with the queryWSSDistributedCacheConfig and queryWSSDistributedCacheCustomConfig commands. There are no required or optional parameters for the query commands.

To list all non-custom configuration settings, run the following Jython command:

```
AdminTask.queryWSSDistributedCacheConfig()
```

To list all distributed cache custom properties, enter the following Jython command:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

- Update your secure conversation distributed cache configuration settings and custom properties.

  Use the following steps to update all non-custom distributed cache configuration settings:

  1. Review your existing configuration settings by running the queryWSSDistributedCacheConfig command, as the following example demonstrates:

```
AdminTask.queryWSSDistributedCacheConfig()
```

     The command returns a properties object that contains the configuration properties and values for the distributed cache configuration. The following table displays the configuration properties that the command returns:

Table 88. Returned configuration properties. Use the properties to determine the distributed cache configuration.

Property	Description
tokenRecovery	Specifies whether token recovery is enabled or disabled. If the tokenRecovery property is set to `true`, the Datasource property specifies the shared data source that is assigned to the distributed cache.
distributedCache	Specifies whether distributed caching is enabled or disabled.
Datasource	Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled.
renewIntervalBeforeTimeoutMinutes	Specifies the amount of time, in minutes, that the client waits before it attempts to renew the token.
synchronousClusterUpdate	Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled.
minutesInCacheAfterTimeout	Specifies the amount of time that the token remains in the cache after the token times out.

  2. Use the updateWSSDistributedCacheConfig command to enable or disable distributed cache and to modify the amount of time after token expiration when downstream calls are allowed to complete.

     The following command example enables distributed cache, and sets the `mySharedDataSource` as the shared data source for token recovery:

```
AdminTask.updateWSSDistributedCacheConfig('[-tokenRecovery true -Datasource
 mySharedDataSource -distributedCache true')
```

  3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

  Use the following steps to update custom properties for your distributed cache configuration:

  1. Review your existing configuration settings by executing the queryWSSDistributedCacheCustomConfig command. For example:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

     The command returns a properties object that contains the name and value pairs that correspond to each custom property.

  2. Use the updateWSSDistributedCacheCustomConfig command to add custom properties for your distributed cache configuration. Specify and define each custom property by passing a properties object with the -customProperties parameter using the following Jython format:

```
-customProperties [[property1 value1][property2 value2]]
```

     For example, the following command adds the `cancelActionRST` custom property and defines the value as `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel`:

```
AdminTask.updateWSSDistributedCacheCustomConfig('[-customProperties [[cancelActionRST
 http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel]]]')
```

3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

- Remove custom properties from your distributed cache configuration. Use the following steps to remove custom properties from your distributed cache configuration:
  1. Review your existing configuration settings by executing the queryWSSDistributedCacheCustomConfig command. For example:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

  2. Use the deleteWSSDistributedCacheConfigCustomProperties command to remove custom properties for your distributed cache configuration. Specify the custom properties to delete by passing a string array with the -propertyNames parameter. For example, the following command removes the `cancelActionRST` custom property:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('[-propertyNames
 [cancelActionRST]]')
```

  3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

Results

Your WS-Security distributed cache configuration is updated.

Configuring custom policies and bindings for security tokens using wsadmin scripting

Use the setPolicyType and setBinding commands for the AdminTask object to specify security tokens for custom policy and binding configurations.

Before you begin

Create a new custom policy set.

About this task

The following scenarios configure the custom policy and bindings to use a Kerberos token based on the Oasis Kerberos Token Profile V1.1 specification. You can also use the setPolicyType and setBinding commands to configure other binary security tokens, such as username tokens, Lightweight Third-Party Authentication (LTPA) and SecureConversation.

Procedure

- Configure custom policies for security tokens.
  1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
  2. Display the properties of the policy of interest.

     Use the getPolicyType command to display detailed property information for the WS-Security policy type, as the following command demonstrates:

```
AdminTask.getPolicyType('-policySet AuthenticationTokenService -policyType
 WSSecurity')
```

     The getPolicyType command returns a properties object that contains name and value pairs for each property, as the following sample output displays:

```
'[ [SupportingTokens.request:krb_token.CustomToken.IncludeToken
 http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient] [enabled true] [type WSSecurity]
 [description [Policies for sending security tokens and providing message confidentiality and integrity, based on the OASIS Web
 Service Security and Token Profiles specifications.]] [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri ]
 [provides ] [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ] ]'
```

  3. Specify the authentication token for the policy type.

Use the setPolicyType command to specify the Uniform Resource Identifier (URI) of the authentication token for services as the value for the `SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri` property. Use the `[ ]` syntax to specify an empty string. The following example specifies an empty string as the value for the authentication token:

```
AdminTask.setPolicyType('-policySet AuthenticationTokenService -policyType
 WSSecurity -attributes "[ [SupportingTokens.request:krb_token.CustomToken.IncludeToken
 http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient] [enabled true] [type
 WSSecurity] [description  [Policies for sending security tokens and providing message confidentiality and integrity,
 based on the OASIS Web Services Security and Token Profiles specifications.]]
 [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri []] [provides []]
 [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ] ]"')
```

- Configure custom bindings for security tokens.
  1. Start the wsadmin scripting tool.
  2. Display the properties of the bindings of interest.

     Use the getBinding command to display detailed property information for the binding of interest, as the following command demonstrates:

```
AdminTask.getBinding('-policyType WSSecurity -bindingLocation "" -bindingName
 AuthenticationTokenService')
```

     The getBinding command returns a properties object that contains name and value pairs for each property, as the following sample output displays:

```
'[ [application.securityinboundbindingconfig.tokenconsumer_0.properties_0.name
 com.ibm.wsspi.wssecurity.krbtoken.serviceSPN]  [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri ]
 [application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
 com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler]  [application.name
 application][application.securityinboundbindingconfig.tokenconsumer_0.properties_0.value HTTP/derekho1.firehorse.austin.ibm.com]
 [application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname system.wss.consume.KRB5BST]
 [application.securityinboundbindingconfig.tokenconsumer_0.name
 con_krbtoken][application.securityinboundbindingconfig.tokenconsumer_0.classname
 com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer]
 [application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference request:krb_token] ]'
```

  3. Specify the authentication token for the policy type.

     Use the setBinding command to specify the Uniform Resource Identifier (URI) of the authentication token for services as the value for the `application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri` property. Use the `[ ]` syntax to specify an empty string. The following example specifies an empty string as the value for the authentication token:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation ""
 -bindingName AuthenticationTokenService -attributes  "[
 [application.securityinboundbindingconfig.tokenconsumer_0.properties_0.name com.ibm.wsspi.wssecurity.krbtoken.serviceSPN]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri []]
 [application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
 com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler] [application.name
 application][application.securityinboundbindingconfig.tokenconsumer_0.properties_0.value
 HTTP/derekho1.firehorse.austin.ibm.com] [application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname
 system.wss.consume.KRB5BST][application.securityinboundbindingconfig.tokenconsumer_0.name
 con_krbtoken][application.securityinboundbindingconfig.tokenconsumer_0.classname
 com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer]
 [application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference request:krb_token]
 ]"')
```

Results

If the setPolicyType and setBinding commands return a `'true'` value, the system successfully updated the policy and binding configurations.

Creating policy set attachments using the wsadmin tool

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your web services applications.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 89. Administrative roles. The administrative role determines if you can create policy set attachments.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can create policy set attachments for application resources only.
Operator	The Operator role cannot create policy set attachments.
Monitor	The Monitor role cannot create policy set attachments.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

About this task

To use a new policy set to manage policies for your application, you must attach the policy set to an application artifact or artifacts. When the application restarts, the application uses the policies from the newly attached policy set.

Note: In a mixed cell environment, the following limitations apply to service reference attachments or resource attachments that are specified in name-value pair format:

- You must not create these types of attachments for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. Service reference attachments are only supported on WebSphere Application Server V8 and later.
- An application that contains these types of attachments must not be deployed on an application server that is prior to WebSphere Application Server Version 8.
- If an application that is deployed in a cluster environment contains these types of attachments, you must not add a member application server that is prior to WebSphere Application Server Version 8 to the cluster.

Procedure

1. Launch a scripting command. To learn more, read about starting the wsadmin scripting client.
2. Select an application with web services to update. Use the listWebServices command to list all web services and the associated applications. Enter the following command to list all web services and attributes:

```
AdminTask.listWebServices()
```

   For each web service, the command returns the associated application name, module name, service name, and service type. For example, the following information is returned:

```
'[ [service {http://www.ibm.com}service1] [client false] [application  application1]
 [module webapp1.war] [type JAX-WS] ]'
```

3. Create a policy set attachment for an application.

For the commands in the PolicySetManagement group, the term **resource** refers to a web service artifact. For application and service client policy sets, the artifacts use the application hierarchy. The application hierarchy includes a web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter.

Note: When attempting to connect to a web service from a thin client, verify that the resources you are specifying are valid before running the updatePolicySetAttachment command. No configuration changes are made if the requested resource does not match a resource in the attachment file for the application.

Use the following format for application and client policy set attachments:

- `WebService:/`

  Attaches all artifacts in the application to the policy set.
- `WebService:/webapp1.war:{http://www.ibm.com}myService`

  Attaches all artifacts within the web service `{http://www.ibm.com}myService` to the policy set. You must provide a fully qualified name (QName) for the service.
- `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA`

  Attaches all operations for the `endpointA` endpoint to the policy set.
- `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1`

  Attaches only the `operation1` operation to the policy set.

The format for the -resource string differs for service reference attachments. Use the following format for service reference attachments:

- `type=WebService:/`

  Attaches all artifacts in the application to the policy set.
- `type=WebService:/,module=myModule.war,service={ http://www.mynamespace.com}myService`

  Attaches all artifacts within the web service `{http://www.mynamespace.com}myService` to the policy set. You must provide a fully qualified name (QName) for the service.
- `type=WebService:/,module=myModule.war,service={ http://www.mynamespace.com }myService,serviceRef=myServiceRef`

  Attaches all artifacts within the web service reference `myServiceRef` to the policy set.
- `type=WebService:/ ,module=myModule.war,service={namespace}myService,serviceRef=myServiceRef,endpoint=endpointA`

  Attaches all operations for the service reference `endpointA` endpoint in the service reference `myServiceRef` to the policy set.
- `type=WebService:/ ,module=myModule.war,service={namespace}myService,serviceRef=myServiceRef,endpoint=endpointA operation=operation1`

  Attaches only the `operation1` operation in the service reference `myServiceRef` to the policy set.

The format for the -resource string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

- `Trust.opName:/`

  The `opName` attribute can be issue, renew, cancel, or validate.
- `Trust.opName:/url`

  The `opName` attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the `url` attribute.

a. Enter the command to attach the policy set to the application. This command attaches the `policyset1` application policy set to all artifacts in the `WebService` application.

   Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for

more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the "`[systemType trustService]`" value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

To attach a policy set to a Web service application, specify the `provider` value for the -attachmentType parameter:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -applicationName WebService -attachmentType provider]')
```

To attach a policy set to a service client application, specify the `client` value for the -attachmentType parameter, as the following example demonstrates:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -applicationName WebService -attachmentType client]')
```

To create a trust service attachment for a system policy set, specify the `provider` value for the -attachmentType parameter and the `[systemType trustService]` value for the -attachmentProperties parameter, as the following example demonstrates:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -attachmentType provider -attachmentProperties "[systemType trustService]"]')
```

To attach a policy set to a service reference, enter the following command:

```
AdminTask.createPolicySetAttachment('[-resources "type=WebService:/,module=webapp1.war,service=
{http://www.mynamespace.com}myService,serviceRef=myServiceRef" -applicationName application1
–attachmentType client -policySet PolicySet1 –inheritFromService false]')
```

This command returns an attachment ID number that you must use to reference this attachment. In the next step, use the attachment ID number to set the binding configuration. For this example, the attachment ID number is `124`.

4. Run the command to set the binding.

   To attach a policy set to a web services application, specify the `provider` value for the -attachmentType parameter.

   The following example demonstrates how to set the timestamp expiration attribute on the `SecureConversation123binding` binding for the `WSSecurity` policy, on the `WebService` Web service application.

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[[application WebService]
[attachmentId 124] ]" -attachmentType provider
 -bindingName SecureConversation123binding -attributes
"[application.securityoutboundbindingconfig.timestampexpires.expires 5]"')
```

   To attach a policy set to a Web services application client or to a service reference, specify the `client` value for the -attachmentType parameter.

5. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

Your have attached the policy set to the application artifact or artifacts specified. Restart your application to use the policies from the newly attached policy set.

What to do next

Manage and update your attachments.

Listing policy sets available for attachment using the wsadmin tool

Use the wsadmin tool to list the supported policy sets that are available to attach to a web services resource. You can attach policy sets to an application, web service, endpoint, or specific operation.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 90. Administrative roles. The administrative role determines if you can create policy set attachments.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can create policy set attachments for application resources only.
Operator	The Operator role cannot create policy set attachments.
Monitor	The Monitor role cannot create policy set attachments.

About this task

To use a new policy set to manage policies for your Web services application, you must attach the policy set to an asset or resource. When the application restarts, the application uses the policies from the newly attached policy set. The listSupportedPolicySets command lists the policy sets which are supported and can be attached to an application or other web services resource.

Procedure

1. Launch a scripting command.
2. Run the command to list supported policy sets. The following example demonstrates how to use the -assetProps parameter to display policy sets that can be attached to a specific type of resource.

   List policy sets that can be attached to an application:

```
AdminTask.listSupportedPolicySets ('[-assetProps [[application myApplication]]]')
```

   List policy sets that can be attached to a WS-Notification service client:

```
AdminTask.listSupportedPolicySets ('[-assetProps [[bus bus1] [WSNService service1]]]')
```

   List policy sets that can be attached to a trust service resource:

```
AdminTask.listSupportedPolicySets ('[-assetProps [[systemType trustService]]]')
```

   List policy sets that can be attached to a business-level application resource. You must specify the business-level application (BLA) name and composition unit (CU) name. Two other properties, BLA edition and CU edition, are optional.

```
AdminTask.listSupportedPolicySets ('[-assetProps [[blaName myBLA] [cuName
 compositionUnit1]]]')
```

   This example uses the optional properties:

```
AdminTask.listSupportedPolicySets ('[-assetProps [[blaName myBLA] [cuName
 compositionUnit1] [blaEdition BASE] [cuEdition 1.0]]]')
```

Results

The command output is a list of policy set names. Use this list to determine which policy sets to attach to your web services resources.

What to do next

Manage and update your policy set attachments. For more information, read about managing policy set attachments using the wsadmin tool.

Managing policy set attachments using the wsadmin tool

Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 91. Administrative roles. The administrative role determines if you can manage policy set attachments.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can manage policy set attachments for application resources only.
Operator	The Operator role cannot manage policy set attachments.
Monitor	The Monitor role cannot manage policy set attachments.

About this task

Policy set attachments define how a policy set is attached to resources and binding configurations.

Procedure

- Query the configuration for policy set attachments and attachment properties.

  Before making configuration changes to your policy set attachments, use the listAttachmentsForPolicySet and getPolicySetAttachments commands to view current configuration information about your policy set attachments.

  1. Start the wsadmin scripting tool.
  2. Use the listAttachmentsForPolicySet command to view all applications to which a specific policy set is attached, for example:

```
AdminTask.listAttachmentsForPolicySet('[-policySet PolicySet1]')
```

     Use the -attachmentType parameter to narrow your query. You can query for `provider` or `client` attachments.

     Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter. For a trust client attachment, or a WSNClient attachment, specify the `client` value for the attachmentType parameter.

3. Use the getPolicySetAttachments command to view the properties for all policy set attachments in a specified application, for example:

```
AdminTask.getPolicySetAttachments('[-applicationName application1]')
```

   Use the -attachmentType parameter to narrow your query. You can query for `provider` or `client` attachments.

- Determine the assets to which a specific policy set is attached.

  Use the listAssetsAttachedToPolicySet command to display the assets that are attached to the policy set of interest, as the following example demonstrates:

```
AdminTask.listAssetsAttachedToPolicySet('[-policySet SecureConversation]')
```

  The command returns a list of properties that describe each asset. Each properties object contains the assetType property, which specifies the type of asset.

- Modify resources that apply to a policy set attachment.
  1. Start the wsadmin scripting tool.
  2. Determine the resource of interest and review the command syntax for the updatePolicySetAttachment command.

     For the commands in the PolicySetManagement group, the term **resource** refers to a web service artifact. For application and service client policy sets, the artifacts use the application hierarchy. The application hierarchy includes a web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter.

     Note: When attempting to connect to a web service from a thin client, verify that the resources you are specifying are valid before running the updatePolicySetAttachment command. No configuration changes are made if the requested resource does not match a resource in the attachment file for the application.

     Use the following format for application and client policy set attachments:

     – `WebService:/`

       Attaches all artifacts in the application to the policy set.

     – `WebService:/webapp1.war:{http://www.ibm.com}myService`

       Attaches all artifacts within the web service `{http://www.ibm.com}myService` to the policy set. You must provide a fully qualified name (QName) for the service.

     – `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA`

       Attaches all operations for the `endpointA` endpoint to the policy set.

     – `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1`

       Attaches only the `operation1` operation to the policy set.

     The format for the -resource string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

     – `Trust.opName:/`

       The `opName` attribute can be issue, renew, cancel, or validate.

     – `Trust.opName:/url`

       The `opName` attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the `url` attribute.

  3. Modify the attachment.

     For example, the policy set attachment is connected to the operation1 operation, which is a specific single operation. To attach the 124 attachment to all operations for the endpointA endpoint, enter the following command:

```
AdminTask.updatePolicySetAttachment('[-attachmentId 124 -resources
"WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA" -applicationName application1]')
```

     Note: The updatePolicySetAttachment command replaces all existing resources for an attachment with the resources specified in the command. You can also update your policy set

attachments using the addToPolicySetAttachment command to add resources to an existing attachment, or you can also use the createPolicySetAttachment command to create an attachment for a specific resource. For more information about these commands reference the commands for the PolicySetManagement group for the AdminTask object.

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

- Remove resources that apply to a policy set attachment.
  1. Start the wsadmin scripting tool.
  2. Determine which resources to remove with the command. You can remove a resource for each web service artifact, each operation for an endpoint, or for a specific operation. In the following example, the command removes the `newAttach` attachment from `operation1`, which is associated with the `plantShop` application.

```
AdminTask.removeFromPolicySetAttachment('[-attachmentId newAttach -resources
"WebService:/webapp1.war:{http://www.ibm.com}myPlantService/endpointA/operation1" -applicationName
plantShop]')
```

     The command returns a success or failure message.

  3. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

- Transfer attachments from one policy set to another policy set. This command detaches each web service from the source policy set and attaches those web services to the destination policy set. The destination policy set must have the same set of enabled policy types as the source policy set.
  1. Enter the following command to transfer all attachments:

```
AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet PolicySet1
-destinationPolicySet PolicySet2]')
```

     The command returns a success or failure message.

  2. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

Managing policy set attachments for service references using the wsadmin tool

Use the wsadmin tool to manage your policy set attachment configurations for services and service references. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 92. Administrative roles. The administrative role determines if you can manage policy set attachments.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource-specific access can manage policy set attachments for application resources only.
Operator	The Operator role cannot manage policy set attachments.
Monitor	The Monitor role cannot manage policy set attachments.

About this task

Policy set attachments define how a policy set is attached to resources and binding configurations.

Application components including application clients, web modules, and EJB modules, can define references to external web services by using logical names called web service references. Before Version 8, web service references always inherited the policy set attachment of the web service.

Note: In Version 8, you can specify a policy set and binding for a service reference that is different from the policy set attachment for the service. In addition, you can indicate to not attach a policy set to a service reference, even if a policy set is attached to the service. The default behavior is that a service reference, as well as its endpoints and operations, inherits the policy set attachment of the corresponding resources of the service. Service references are only valid for the client attachment type.

Note: In a mixed cell environment, the following limitations apply to service reference attachments or resource attachments that are specified in name-value pair format:

- You must not create these types of attachments for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. Service reference attachments are only supported on WebSphere Application Server V8 and later.
- An application that contains these types of attachments must not be deployed on an application server that is prior to WebSphere Application Server Version 8.
- If an application that is deployed in a cluster environment contains these types of attachments, you must not add a member application server that is prior to WebSphere Application Server Version 8 to the cluster.

Procedure

- Query the configuration for policy set attachments and attachment properties.

  Before making configuration changes to your policy set attachments, use the listAttachmentsForPolicySet and getPolicySetAttachments commands to view current configuration information about your policy set attachments.

  1. Start the wsadmin scripting tool.
  2. Use the listAttachmentsForPolicySet command to view all applications to which a specific policy set is attached; for example:

```
AdminTask.listAttachmentsForPolicySet('[-policySet PolicySet1 –attachmentType client]')
```

  3. Use the getPolicySetAttachments command to view the properties for all policy set attachments in a specified application; for example:

```
AdminTask.getPolicySetAttachments('[-applicationName application1 –attachmentType client]')
```

- Determine the assets to which a specific policy set is attached.

  Use the listAssetsAttachedToPolicySet command to display the assets that are attached to the policy set of interest, as the following example demonstrates:

```
AdminTask.listAssetsAttachedToPolicySet('[-policySet SecureConversation]')
```

  The command returns a list of properties that describe each asset. Each properties object contains the assetType property, which specifies the type of asset.

- Use the createPolicySetAttachment command to create the policy set attachment.

  For the commands in the PolicySetManagement group, the term, *resource*, refers to a web service artifact. For service references, use name-value pairs to specify the resource name. You can also use this format instead of the application hierarchy format to specify service resources.

  The following format for application and client policy set attachments is required for service reference resources. For resources that are not service references, you can use either this format or the format

specified in the managing policy set attachments using the wsadmin tool documentation. Use the following format for application and client policy set attachments:

– `type=WebService:/`

  Attaches all artifacts in the application to the policy set.

– `type=WebService:/,module=myModule.war,service={ http://www.mynamespace.com}myService`

  Attaches all artifacts within the web service, `{http://www.mynamespace.com}myService`, to the policy set. You must provide a fully qualified name (QName) for the service.

– `type=WebService:/,module=myModule.war,service={ http://www.mynamespace.com }myService,serviceRef=myServiceRef`

  Attaches all artifacts within the web service reference, `myServiceRef`, to the policy set.

– `type=WebService:/ ,module=myModule.war,service={namespace}myService,serviceRef=myServiceRef,endpoint=endpointA`

  Attaches all operations for the service reference, `endpointA`, endpoint in the service reference, `myServiceRef`, to the policy set.

– `type=WebService:/ ,module=myModule.war,service={namespace}myService,serviceRef=myServiceRef, endpoint=endpointA,operation=operation1`

  Attaches only the `operation1` operation in the service reference,

  `myServiceRef`, to the policy set.

Use the inheritFromService parameter and specify either `true` or `false` to indicate whether you want to inherit the policy set attachment from the service.

– If you specify a service reference resource with `inheritFromService=false` and no `policySet` parameter, all attachments for the service reference are removed and the service reference does not inherit the policy set attachments of the service.

– If you specify a service reference resource with `inheritFromService=false` and `policySet=<policySetName>`, an attachment for the service reference resource to the policy set is created. If you create an attachment to attach a service reference to a policy set, the service reference does not inherit the policy set of the service.

– If you specify a service reference resource with `inheritFromService=true`, the service reference inherits the attachments for the service. The inheritFromService parameter is only valid for service references. If no policySet parameter is specified, the inheritFromService parameter is only valid at the `serviceRef` level, not at the endpoint or operation level.

  For example, to attach policy set `PolicySet1` to all operations and endpoints for the service reference, enter the following command:

```
AdminTask.createPolicySetAttachment('[-resources "type=WebService:/,module=webapp1.war,service=
{http://www.mynamespace.com}myService,serviceRef=myServiceRef" -applicationName application1 –attachmentType
client -policySet PolicySet1 –inheritFromService false]')
```

  To specify that the service reference does not have a policy set attachment and does not inherit the policy set from the service, enter the following command:

```
AdminTask.createPolicySetAttachment('[-resources "type=WebService:/,module=webapp1.war,service=
{http://www.mynamespace.com}myService,serviceRef=myServiceRef" -applicationName application1 –attachmentType
client –inheritFromService false]')
```

- Modify resources that apply to a policy set attachment.
  1. Start the wsadmin scripting tool.
  2. Determine the resource of interest, and review the command syntax for the updatePolicySetAttachment command.
  3. Modify the attachment.

     For example, the policy set attachment is connected to the `operation1` operation, which is a specific single operation. To attach the `124` attachment to all operations for the `endpointA` endpoint in the service reference, enter the following command:

```
AdminTask.updatePolicySetAttachment('[-attachmentId 124 -resources
"type=WebService:/,module=webapp1.war,service={http://www.mynamespace.com}myService,serviceRef=myServiceRef,
endpoint=endpointA" -applicationName application1 –attachmentType client]')
```

The updatePolicySetAttachment command replaces all existing resources for an attachment with the resources that are specified in the command. If you update an attachment to attach a service reference to a policy set, the service reference does not inherit the policy set of the service. You can also update your policy set attachments using the addToPolicySetAttachment command to add resources to an existing attachment, or you can also use the createPolicySetAttachment command to create an attachment for a specific resource. For more information about these commands reference the commands for the PolicySetManagement group for the AdminTask object. A resource for a service reference cannot be included in the same attachment as a resource for a service.

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

- Remove resources that apply to a policy set attachment.
  1. Start the wsadmin scripting tool.
  2. Determine which resources to remove with the command. You can remove a resource for each web service artifact, each endpoint operation, or for a specific operation. In the following example, the command removes the `newAttach` attachment from `operation1` in the service reference `myPlantServiceRef`, which is associated with the `plantShop` application and `myPlantService` service:

```
 AdminTask.removeFromPolicySetAttachment('[-attachmentId newAttach -resources
"type=WebService:/,module=webapp1.war,service={http://www.mynamespace.com}myPlantService,
serviceRef=myPlantServiceRef,endpoint=endpointA,operation=operation1" -applicationName plantShop
–attachmentType client]')
```

     The command returns a success or failure message.

  3. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

- Transfer attachments from one policy set to another policy set. This command detaches each web service from the source policy set and attaches those web services to the destination policy set. The destination policy set must have the same set of enabled policy types as the source policy set.
  1. Enter the following command to transfer all attachments:

```
AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet PolicySet1
-destinationPolicySet PolicySet2]')
```

     The command returns a success or failure message.

  2. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

Configuring general, cell-wide bindings for policies using wsadmin scripting

You can use the Jython or Jacl scripting language to customize your cell-wide default binding configuration. Create multiple cell-wide general bindings that you can attach to applications.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 93. Administrative roles. The administrative role determines if you can configure or assign bindings.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Bindings are environment and platform-specific information such as key store information, keys used for signature and encryption, or authentication information.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 and trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 and later bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 and later environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify

default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

Procedure

1. Start the wsadmin scripting tool.
2. Determine the policy to update.

   To view a list of all available policies for a specific policy set, use the listPolicyType command. For example:

```
AdminTask.listPolicyTypes('[-policySet PolicySet1]')
```

3. Retrieve the current binding configuration for the policy to determine the attributes to update.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific policy binding. Specify a Properties object for the -bindingLocation parameter using an empty Properties object. For example:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1')
```

   To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if the WSAddressing policy has workload management enabled:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1 -attributes "[preventWLM]"')
```

   The command returns a properties object which contains the value of the requested attribute, preventWLM.

4. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a cell-wide binding, set the -bindingLocation parameter by passing a null or empty Properties object and specify the name of the binding with the -bindingName parameter. You can further customize your binding with the following parameters:

Table 94. Command parameters. Use the command to update the binding configuration.

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update.	Properties, optional
-replace	Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is false.	Boolean, optional
-remove	Use this parameter to remove a specific policy from the binding configuration. The default value for the remove parameter is false. If the policyType parameter is not specified, the command removes the custom binding from the attachment. To delete the binding configuration, provide a value for the bindingName parameter and an asterisk character (*) for the attachmentId.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

You must use the -attributes parameter when editing your binding configuration for cell-wide bindings. The following example disables workload management within the cell-wide default binding for the `WSAddressing` policy:

```
AdminTask.setBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1 -attributes "[preventWLM false]"')
```

5. Save your configuration changes.

```
AdminConfig.save()
```

Configuring Version 6.1 server-specific default bindings for policies using wsadmin scripting

You can use the Jython or Jacl scripting language to customize WebSphere Application Server Version 6.1 server-specific default bindings for policies to match your installation environment or requirements.

Before you begin

Server-specific default bindings use the WebSphere Application Server Version 6.1 namespace.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 95. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 and trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.

- To use or modify general Version 7.0 and later bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 and later environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
2. Determine the policy to update.

   To view a list of all available policies for a specific policy set, use the listPolicyTypes command, as the following example demonstrates:

```
AdminTask.listPolicyTypes('[-policySet WSAddressing]')
```

3. Retrieve the current binding configuration for the policy to determine the attributes to update.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific policy binding. Specify a Properties object for the -bindingLocation parameter using the property names `node` and `server`. For example:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[node node1]
[server server1]]"')
```

   To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if the policy is enabled:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[node node1]
[server server1]]" -attributes "[[preventWLM]]"')
```

The command returns a properties object which contains the value of the requested attribute, `preventWLM`.

4. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a server-specific default binding, set the -bindingLocation parameter using the `node` and `server` property names in a Properties object. You can further customize your binding with the following optional parameters:

Table 96. Optional parameters. Use the following optional parameters to customize bindings.

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-remove	Use this parameter to remove a server-level binding configuration. The default value for the -remove parameter is `false`.	Boolean, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update. The -attributes parameter is not required if you are removing your server-level binding.	Properties, optional
-replace	Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is `false`.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

   You should always specify the -attributes parameter when editing your configuration. The following example disables workload management within the server-specific default binding for the `WSAddressing` policy:

```
AdminTask.setBinding('-policyType WSAddressing -bindingLocation "[ [server server1] [node node01] ]" -attributes "[preventWLM false]"')
```

5. Save your configuration changes.

```
AdminConfig.save()
```

Configuring application-specific and system bindings using wsadmin scripting

Use the Jython or Jacl scripting language to edit custom application bindings and system bindings for policies to match your installation environment or system requirements.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 97. Administrative roles. The administrative role determines if you can configure, modify or assign bindings.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.

Table 97. Administrative roles (continued). The administrative role determines if you can configure, modify or assign bindings.

Administrative role	Authorization
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Binding configurations are environment- and platform-specific information such as keystore information, keys used for signature and encryption, or authentication information. You can use the default binding for each policy set or define application-specific bindings within an application.

There are three types of bindings to use with your policy sets, including cell-level, application server level, and application-level. Default bindings are used at the cell-level or application server level. This topic refers to system binding information or bindings that are defined at the application level, which overrides the cell-level or application server level definition.

Use default bindings only to develop and test applications. You must change signing and encryption keys before using your bindings in a production environment.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 and trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 and later bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 and later environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
2. Retrieve the current binding data for the attachment of interest.

   Use the getPolicySetAttachments command to determine the attachment ID. You will need to specify the attachment ID in the getBinding and setBinding commands to specify that this is a application-specific binding configuration. Use the following command to retrieve the attachment ID:

```
AdminTask.getPolicySetAttachments('-applicationName application1')
```

   Use the getBinding command to display a properties object that contains each configuration attribute for a specific policy binding configuration. For application and client policy set attachments, specify a properties object for the -bindingLocation parameter using the `application` and `attachmentId` property names. For a system policy set attachment for the trust service, specify only the `attachmentId` property name. The following example queries for an application policy set binding configuration:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId
123]]"')
```

   To return a specific configuration attribute for the policy, use the -attributes parameter.
3. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a application-specific binding configuration, set the -bindingLocation parameter by specifying the `application` and `attachmentId` property names in a properties object. You can additionally specify the -attachmentType parameter as `provider` or `client`.

   Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

   Customize your binding configuration with the following optional parameters:

Table 98. Optional parameters. Use the optional binding parameters to update the binding configuration.

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-remove	Use this parameter to remove a specific policy from the binding configuration. The default value for the -remove parameter is false. If the -policyType parameter is not specified, the command removes the application-specific binding from the attachment. To delete the binding configuration, provide a value for the -bindingName parameter and an asterisk character (*) for the -attachmentId parameter.	Boolean, optional
-attributes	Specifies the attribute values to update. This parameter can include each binding configuration attribute for the policy or a subset of attributes to update. If you do not specify the attributes parameter, the command only updates the binding configuration location that the specified attachment uses.	Properties, optional
-bindingName	Specifies the name for the binding configuration. Use this parameter to specify a name for the binding when you create a new application-specific binding. You can also use this parameter to switch an attachment to use a different, existing application-specific binding configuration. Lastly, you must specify a value for this parameter to delete a binding configuration.	String, optional
-replace	Specifies whether to replace all of the existing binding configuration attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is `false`.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

The following example disables workload management for the `myApplication` application's binding configuration for the `WSAddressing` policy:

```
AdminTask.setBinding('[-policyType WSAddressing -bindingLocation "[ [application myApplication]
[attachmentId 123] ]"
 -attributes "[preventWLM false]" -attachmentType provider]')
```

4. Save the configuration changes.

   Enter the following command to save your changes.

   ```
   AdminConfig.save()
   ```

Creating application-specific and trust service-specific bindings using the wsadmin tool

You can use the Jython or Jacl scripting language to create application-specific and trust service-specific bindings to match your installation environment or requirements.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 99. Administrative roles. The administrative role determines if you can configure or assign bindings.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can configure binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.

Table 99. Administrative roles (continued). The administrative role determines if you can configure or assign bindings.

Administrative role	Authorization
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Policy set bindings specify the details about how your quality of service (QoS) is configured. For example, a policy set attachment determines that sign, encrypt, or reliable messaging should be enabled. The policy set binding specifies how the protection is configured, for example, the path of the keystore file, the class name of the token generator, or the Java™ Authentication and Authorization Service (JAAS) configuration name.

For application policy sets, you can specify the policy set bindings at the cell-level using default binding configurations, at the application level using application-specific binding configurations, or at the cell-level with general bindings. Server-level default bindings are deprecated. If no binding information is specified during policy set attachment, the policy set inherits the default binding. You can specify a general binding as the default for a server instead of server-default bindings.

For system policy sets, you can specify the bindings at the cell-level and the server-level. The available bindings for system policy sets are the TrustServiceSymmetricDefault and TrustServiceSecurityDefault bindings. If no custom binding information is specified by the attachment, the resources inherit the TrustServiceSymmetricDefault or TrustServiceSecurityDefault binding.

Note: Only use default binding for development and testing. You must customize the signing and encryption keys in your binding configurations for a production environment.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 and trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 and later bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 and later environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Determine the type of binding to create.

   You can create application policy set bindings at the cell-level, server-level, or application-level, and trust service policy set bindings at the cell-level or server-level.
3. Retrieve the current binding configuration for the policy of interest.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific binding. Specify the location of the binding by passing a properties object using the -bindingLocation parameter and the following reference table:

Table 100. Command parameters. Use the command to display attributes for a binding.

Type of binding	-bindingLocation parameter value
Server-level (deprecated)	`-bindingLocation "[[node node1][server server1]]"`
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service	`-bindingLocation "[[systemType trustService] [attachmentId 123]]"`
Trust client	`-bindingLocation "[[systemType trustClient] [attachmentId 123]]"`
WS-Notification client	`-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"`

For this example, the command displays the current binding configuration for the `WSAddressing` policy, with the `123` attachment ID, for the `application1` application:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 123]]"')
```

To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if workload management is enabled:

```
AdminTask.getBinding('-policyType WSaddressing -bindingLocation "[[application application1][attachmentId 123]]" -attributes "[preventWLM]"')
```

The command returns a properties object which contains the value of the requested attribute, `preventWLM`. You might receive an error message if the binding does not exist in your configuration.

4. Create a new application-specific binding for the policy of interest.

   Use the setBinding command to create a binding configuration for a policy. To specify that you are creating an application-specific binding, set the -bindingLocation parameter by passing the `application` and `attachmentId` property names in a properties object. If you are creating a system policy set binding for the trust service, you only need to specify the `attachmentId` property name. You can further customize your binding with the following parameters:

Table 101. Command parameters. Use the command to create a binding configuration.

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-attachmentType	Specifies the type of policy set attachment. If the attachment is for an application, you do not need to specify this parameter. **Note:** Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.	String, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset of attributes.	Properties, optional
-bindingName	Specifies the name for your new application-specific binding. A name is generated if it is not specified.	String, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

The following example creates the `WSAddressing1234binding` attachment-specific binding for the `WSAddressing` policy, assigned to the `application1` application attachment `123`, and enables workload management:

```
AdminTask.setBinding('-policyType WSAddressing -bindingName
 WSAddressing123binding -bindingLocation "[ [application application1] [attachmentId 123] ]" -attributes
 "[preventWLM false]"')
```

5. Optional: Add application-specific binding properties.

   Use the setBinding command to add any additional custom properties for your application-specific binding. The application server provides custom properties that are specific to each quality of service. Use the following format to specify custom properties for the binding:

```
AdminTask.setBinding('[-bindingLocation "[ [application application1] [attachmentId 123]
 ]" -policyType WSAddressing -attributes "[[properties_x:name key_value] [properties_x:value
 value]"]')
```

6. Save your configuration changes.

```
AdminConfig.save()
```

Deleting application-specific bindings from your configuration using wsadmin scripting

You can use the Jython or Jacl scripting language to delete a custom application or system policy set binding from your configuration. You cannot delete cell-level default bindings.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 102. Administrative roles. The administrative role determines if you can delete or modify bindings.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to modify bindings. If you have access to a specific resource only, you can modify bindings for the resource for which you have access.
Configurator	The Configurator role cannot modify bindings.
Deployer	The Deployer role cannot modify bindings.
Operator	The Operator role cannot modify bindings.
Monitor	The Monitor role cannot modify bindings.

About this task

Policy set bindings specify the details about how your quality of service (QoS) is configured. For example, a policy set attachment determines that sign, encrypt, or reliable messaging is enabled. The policy set binding specifies how the protection is configured, for example, the path of the keystore file, the class name of the token generator, or the Java Authentication and Authorization Service (JAAS) configuration name.

For application policy sets, policy set bindings exist at the cell-level and server-level using default binding configurations, or at the application level using application-specific binding configurations. You can also specify cell-level general bindings. For system policy sets, bindings exist at the cell level and server level, or you can create application-specific bindings.

Use the following procedure to delete application-specific bindings for trust policy sets and application level bindings for application policy sets:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. Retrieve the current binding configuration for the policy of interest.

   Use the getBinding command to display a properties object that contains all configuration attributes for a specific binding. Specify the location of the binding by passing a properties object using the bindingLocation parameter and the following reference table:

Table 103. bindingLocation parameter options. Use the parameter to control the output of the getBinding command.

Type of Binding	Value for the -bindingLocation parameter
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service	`-bindingLocation "[[attachmentId 123]]"`
WS-Notification client	`-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"`
General binding	`-bindingLocation []`

In this example, the command displays the current binding configuration for the WSAddressing policy, with the 123 attachmentId, for the application1 application:

```
AdminTask.getBinding('[-policyType WSAddressing -bindingLocation  "[[application
 application1][attachmentId 123]]"]')
```

To display general policy set bindings, identify the bindings by specifying the -bindingName parameter, as the following example demonstrates:

```
AdminTask.getBinding('[-bindingLocation [] -attachmentType application
 -bindingName "General Provider Binding"]')
```

3. Remove the binding of interest from each attachment.

   You cannot remove a binding from your configuration if that binding is referenced by one or more attachments. Modify and use the following example command to remove a binding from an attachment:

```
AdminTask.setBinding('[-bindingLocation "[[application application1][attachmentId
 123]]"  -remove true]')
```

4. Delete the binding of interest.

   Use the setBinding command to delete a application-specific binding configuration. Specify the binding of interest with the -bindingName parameter, an asterisk (*) for the -attachmentId property, and set the -remove parameter to true. The following example setBinding command removes the WSAddressing123binding application policy set binding:

```
AdminTask.setBinding('[-attachmentType application -bindingName
 WSAddressing123binding  -bindingLocation "[[application application1][attachmentId *]]" -remove
 true]')
```

   The following example setBinding command removes the customTrust trust service binding:

```
AdminTask.setBinding('[-attachmentType "system/trust" -bindingName
 customTrust  -bindingLocation "[attachmentId *]" -remove true]')
```

   The following example setBinding command removes the General Provider Binding general binding:

```
AdminTask.setBinding('[-attachmentType application -bindingName "General Provider
 Binding" -bindingLocation [] -bindingScope domain -remove true]')
```

   Note: You cannot delete general bindings if an attachment references the binding, or if the binding is set as the default for a server or domain.

5. Save your configuration changes.

   Use the following command example to save your configuration changes:

```
AdminConfig.save()
```

Results

The application-specific binding of interest is removed from your configuration.

Importing and exporting policy sets to client or server environments using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 104. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to import and export policy sets.
Configurator	The Configurator role cannot import and export policy sets.

Table 104. Administrative roles (continued). This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Deployer	The Deployer role cannot import and export policy sets.
Operator	The Operator role cannot import and export policy sets.
Monitor	The Monitor role cannot import and export policy sets.

About this task

You can use the exportPolicySet and importPolicySet commands to exchange system or application policy sets between servers or between a client and a provider. To reuse a policy set on a new server or client, export the policy set to an archive file, then import the archive file on the destination server or client. This topic provides examples for exporting a policy set, importing a policy set from an archive file, and importing a default policy set.

Procedure

- Export an application or system policy set to an archive file.

  Use the exportPolicySet command to create an archive file for the policy set of interest. For example, the following command creates the customSC.zip archive file in the C:\IBM\WebSphere\AppServer\PolicySets\ directory for the customSecureConversation policy set:

  ```
  AdminTask.exportPolicySet('[-policySet customSecureConversation
  -pathName C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip]')
  ```

- Move the policy set archive file to the destination environment.

  If you are exporting the policy set to a client environment, then place the archive file on the classpath of the client.

- Import a policy set from an archive file or import a default policy set.

  Use the importPolicySet command to import the archive file containing the policy set configuration of interest to the destination environment. You cannot import a policy set onto a server or client environment if the policy set already exists in the destination environment.

  For example, the following command creates a customSecureConversation policy set from the customSC.zip archive file:

  ```
  AdminTask.importPolicySet('[-importFile C:\IBM\WebSphere\AppServer\bin\customSC.zip]')
  ```

  Additionally, you can also use the importPolicySet command to import a default policy set onto a server environment, as the following example demonstrates:

  ```
  AdminTask.importPolicySet('[-defaultPolicySet SecureConversation -policySet copyOfdefaultSC]')
  ```

- Save the configuration changes.

  Enter the following command to save your changes:

  ```
  AdminConfig.save()
  ```

Removing policy set bindings using wsadmin scripting

You can use the Jython or Jacl scripting language to remove binding configurations for policies and resources to match your installation environment or requirements.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 105. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete bindings. If you have access to a specific resource only, you can delete bindings for the resource for which you have access.
Configurator	The Configurator role can unassign bindings, but cannot delete bindings.
Deployer	The Deployer role can unassign bindings, but cannot delete bindings.
Operator	The Operator role cannot modify bindings.
Monitor	The Monitor role cannot modify bindings.

About this task

Use the following steps to remove specific policies from your application-specific binding configuration, or to remove your entire binding configuration. For both of these removal options, you must use the -bindingLocation parameter to specify whether you are deleting an application-specific binding, server-specific default binding, or a binding for the trust service. Use the following table for examples using Jython syntax when specifying the type of binding to modify or remove:

Table 106. bindingLocation parameter options. Use the following values for the -bindingLocation parameter for the associated types of bindings.

Type of binding	Value for the -bindingLocation parameter
Server-level (for Version 6.1 bindings only)	`-bindingLocation "[[node node1][server server1]]"`
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service binding	`-bindingLocation "[[systemType trustService] [attachmentId 123]]"`
WS-Notification client	`-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"`
General bindings	`-bindingLocation []`

Procedure

- Remove a policy from your application-specific binding configuration.

  Use the following steps to remove a specific policy from your binding configuration. If you remove the last policy remaining in your binding configuration, the command removes binding information from all attachments and deletes it from your configuration.

  1. Launch the wsadmin scripting tool using the Jython scripting language. To learn more, see the starting the wsadmin scripting client information.
  2. Review the binding configuration to edit.

     Use the getBinding command to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 1234]]"')
```

     If the binding of interest is not referenced by an attachment ID, specify an asterisk character (*) for the attachmentId parameter to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId *]]"')
```

  3. Remove the policy from the binding configuration.

Use the setBinding command with the -policyType and -remove parameters to remove the policy of interest from the binding configuration. For example, use the following command to remove the WSAddressing policy from the binding configuration for the application1 application:

```
AdminTask.setBinding('-policyType WSAddressing -remove true -bindingLocation "[[application application1][attachmentId 1234]]"
```

If the binding to delete is not referenced by an attachment ID, specify an asterisk character (*) for the attachmentId parameter to delete the binding, as the following example demonstrates:

```
AdminTask.setBinding('-policyType WSAddressing -remove true -bindingLocation "[[application application1][attachmentId *]]"')
```

4. Save your configuration changes.

   Use the following command example to save your configuration changes:

   ```
   AdminConfig.save()
   ```

- Remove binding configurations from an attachment.

  Use the following steps to remove a server-specific default binding or a custom binding. You cannot remove cell-level default bindings from your configuration. When a binding is removed from an attachment, the resource it was removed from will inherit the server-level default binding, if one is present, or the cell-level default binding if the server-level binding is not present. Use the following steps to remove a binding configuration:

  1. Start the wsadmin scripting tool.
  2. Verify the current binding configuration to delete.

     Before removing the binding from the attachment, use the getBinding command to view the attributes for the binding, as the following example demonstrates:

     ```
     AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 123]]"')
     ```

  3. Remove the current binding configuration from the attachment.

     For this example, this command removes the bindings from the 123 attachment for the application1 application:

     ```
     AdminTask.setBinding('-bindingLocation "[[application application1][attachmentId 123]]" -remove true')
     ```

     If the binding to delete is not referenced by an attachment ID, specify an asterisk character (*) for the -attachmentId parameter to remove the binding, as the following example demonstrates:

     ```
     AdminTask.setBinding('-bindingLocation "[[application application1][attachmentId *]]" -remove true')
     ```

     To remove a server-specific default binding, specify the node name and server name with the -bindingLocation parameter. Server specific default bindings are deprecated. For example, this command removes the server-level default binding for the WS-Addressing policy from the server1 server on the node1 node:

     ```
     AdminTask.setBinding('-policyType WSAddressing -bindingLocation "[[node node1][server server1]]" -remove true')
     ```

  4. Save your configuration changes.

     Use the following command example to save your configuration changes:

     ```
     AdminConfig.save()
     ```

- Remove a policy from a general binding.
  1. Start the wsadmin scripting tool.
  2. Verify the current binding configuration to delete.

     Before removing the binding from the attachment, use the getBinding command to view the attributes for the binding, as the following example demonstrates:

     ```
     AdminTask.getBinding('-policyType WSAddressing -bindingName "General Provider Binding" -bindingLocation []')
     ```

  3. Remove the general binding.

     For this example, this command removes the General Provider Binding general binding:

     ```
     AdminTask.setBinding('-bindingLocation [] -bindingName "General Provider Binding" -remove true')
     ```

  4. Save your configuration changes.

     Use the following command example to save your configuration changes:

     ```
     AdminConfig.save()
     ```

Removing policy set attachments using the wsadmin tool

You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 107. Administrative roles. The administrative role determines if you can remove policy set attachments.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to remove policy set attachments. If you have access to a specific resource only, you can remove policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to remove policy set attachments. If you have access to a specific resource only, you can remove policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can remove policy set attachments for application resources only.
Operator	The Operator role cannot remove policy set attachments.
Monitor	The Monitor role cannot remove policy set attachments.

Determine which applications and policy sets to remove, detach, or transfer. Use the listWebServices command to list all Web services and for the application to edit. Enter the command to list all web services and attributes for a specific application.

```
AdminTask.listWebServices('[–application application_name]')
```

To view a list of all web services and associated applications, do not provide the -application parameter. For each web service, the command returns the associated application name, module name, service name, and service type. You can also use the listAttachmentsForPolicySet and getPolicySetAttachments administrative commands to view existing configuration data. For additional information about these commands, use the information center topic for the PolicySetManagement group of commands for the AdminTask object.

About this task

There are four ways to remove policy set attachments, including:

- Remove a policy set attachment from an application.
- Remove resources that apply to a policy set attachment.
- Remove all attachments for a specific policy set and application.
- Transfer attachments between policy sets for a specific application.

Choose the appropriate procedure to remove your policy set attachments.

Procedure

- Remove a policy set attachment from the application.
  1. Enter the following command to remove the policy set attachment from the application:

```
AdminTask.deletePolicySetAttachment('[-attachmentId attachment_name -applicationName
application_name]')
```

     The command returns a success or failure message. If you receive a success message, the policy set attachment is successfully removed from your application. If you receive a failure message, verify that the chosen policy set attachment exists in your configuration.

2. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

- Remove resources that apply to a policy set attachment.

  1. You can customize the command to remove resources for the web service, endpoint, or operation.

     For the commands in the PolicySetManagement group, the term *resource* refers to a web service artifact. For application and service client policy sets, the artifacts use the application hierarchy: Web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter. Use the following format for application and client policy set attachments:

     WebService:/
     : Refers to all artifacts in the application to the policy set.

     WebService:/webapp1.war:{http://www.ibm.com}myService
     : Refers to all artifacts within the web service *{http://www.ibm.com}myService* to the policy set. You must provide a fully qualified name (QName) for the service.

     WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA
     : Refers to all operations for the *endpointA* endpoint to the policy set.

     WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1
     : Refers to only the *operation1* operation to the policy set.

     The format for the **resource** string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

     Trust.<opName>:/
     : The *<opName>* attribute can be issue, renew, cancel, or validate.

     Trust.<opName>:/url
     : The *<opName>* attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the *url* attribute.

     In the following example, the command removes the `attachment_name` attachment from the `operation1` operation, which is associated with the application, `application1`.

     ```
     AdminTask.removeFromPolicySetAttachment('[-attachmentId attachment_name -resources
     "WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1" -applicationName
     application1]')
     ```

     The command returns a success or failure message. You can also use the **updatePolicySetAttachments** command to remove attached resources.

  2. Save the configuration changes.

     Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```

- Remove all attachments for a specific policy set.

  1. Remove application attachments for the policy set. To remove an attachment from an application, use the following command:

     ```
     AdminTask.deleteAttachmentsForPolicySet('[-policySet PolicySet1 -applicationName
      application1]')
     ```

     To remove all attachments from the policy set, use the following command:

     ```
     AdminTask.deleteAttachmentsForPolicySet('[-policySet PolicySet1]')
     ```

     Both commands return a success or failure message.

  2. Save the configuration changes.

     Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```

- Transfer attachments from one policy set to another policy set. This command detaches all web services from the source policy set and attaches the web services to the destination policy set.
  1. Enter the transferAttachmentsForPolicySet command to transfer all attachments within an application. Use the following command to transfer the attachments from the *PolicySet1* policy set to the *PolicySet2* policy set within the *application1* application:

```
AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet PolicySet1
-destinationPolicySet PolicySet2 -applicationName application1]')
```

     The command returns a success or failure message.
  2. Save the configuration changes.

     Enter this command to save your changes:

     ```
     AdminConfig.save()
     ```

Deleting policy sets using wsadmin scripting

Use the Jython or Jacl scripting language to delete policy sets from your configuration with the wsadmin tool. You must remove all policy set attachments before removing the policy set.

Before you begin

To complete this task, you must use the Administrator role with cell-wide access when administrative security is enabled.

Before deleting a policy set, you must delete or transfer all attachments to applications. You cannot delete default policy sets.

About this task

Use the following steps to delete custom policy sets from your configuration with the wsadmin tool:

Procedure

1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
2. List all policy sets in your configuration.
   - Enter the following command to list all application policy sets:

     ```
     AdminTask.listPolicySets()
     ```
   - Enter the following command to list all policy sets for the trust service:

     ```
     AdminTask.listPolicySets('[-policySetType system/trust]')
     ```
   - Enter the following command to list all system policy sets:

     ```
     AdminTask.listPolicySets('[-policySetType system]')
     ```
3. Determine which policy set to delete. Enter the following command to view the description and default indicator for a specific policy set:

   ```
   AdminTask.getPolicySet('[-policySet policySet_name]')
   ```
4. Delete the policy set.

   Enter the following command to delete a specific policy set.

   ```
   AdminTask.deletePolicySet('[-policySet PolicySet1)')
   ```

   The command returns a success or failure response. If you receive an error message, make sure that you have deleted all policy set attachments before entering the deletePolicySet command.
5. Save the configuration changes.

   Enter the command to save your changes.

   ```
   AdminConfig.save()
   ```

What to do next

If you deleted a policy set that was previously attached to an application, restart the affected application to update the configuration changes.

Refreshing policy set configurations using wsadmin scripting

Use the wsadmin tool to refresh the policy set configuration data. After refreshing the policy set configuration, the changes apply after restarting the application.

Procedure

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Get the object name of each PolicySetManager object.

   Use the completeObjectName option for the AdminControl object to set the object name for each PolicySetManager type object to the `objNameString` variable, as the following example demonstrates:

```
objNameString = AdminControl.completeObjectName('type=PolicySetManager,*')
```

3. Connect to the Managed Bean (MBean).

   The MBean supplies a remote interface to the MBean server that runs in the application server. The following example shows how to look up the MBean:

```
import javax.management as mgmt
```

4. Set the PolicySetManager MBean object name.

   The following example sets the PolicySetManager MBean object name to the `mbeanObj` variable, parameters to the `param` variable, and signature settings to the `sig` variable:

```
mbeanObj = mgmt.ObjectName(objNameString)
param=[]
sig=[]
```

5. Refresh the PolicySetManager MBean.

   The following example refreshes the policy set configuration:

```
AdminControl.invoke_jmx(mbeanObj, 'refresh', param, sig)
```

Example

The following example provides the Jython script that refreshes the policy set configuration:

```
objNameString = AdminControl.completeObjectName('type=PolicySetManager,*')
import javax.management as mgmt
mbeanObj = mgmt.ObjectName(objNameString)
param=[]
sig=[]
AdminControl.invoke_jmx(mbeanObj, 'refresh', param, sig)
```

Policy configuration properties for all policies

You can use the **attributes** parameter with the setPolicyType and setBinding commands to specify various properties for each quality of service (QoS) within a policy set. You can use the properties in this topic with each QoS within application and system policy sets.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Attributes to configure for all QoS policies

Use the following list of attributes to configure attributes across all QoS policies using the Jython scripting language and the wsadmin tool:

enabled

Specifies whether the policy type is enabled or disabled. The following example provides the format to enter the attributes parameter:

```
-attributes "[[enabled true]]"
```

provides

Provides a description for your configuration. The following example provides the format to enter the attributes parameter:

```
-attributes "[[provides [Messaging Security]]]"
```

The following example uses the setPolicyType command to set the `enabled` and `provides` properties for the myCustomSecurityPS custom policy set, which contains a ReliableMessaging policy:

```
AdminTask.setPolicyType('[-policySet myCustomSecurityPS -policyType
WSReliableMessaging -attributes [[enabled true][provides
[Messaging security]]]]')
```

WSSecurity policy and binding properties

Use the **attributes** parameter for the setPolicyType and setBinding commands to specify additional configuration information for the WSSecurity policy and binding configurations. Application and system policy sets can use the WSSecurity policy and binding configuration.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

If the **attributes** parameter is not specified for the getPolicyType or getBinding command, the command returns all properties. If a partial property name is passed to the getPolicyType or getBinding command, the command returns all properties with names that start with the partial property name. For example, If SignatureProtection is passed to the getPolicyType command, the command returns all properties with names that start with "SignatureProtection", which might include:

```
SignatureProtection.response:
   int_body.SignedParts.Body,SignatureProtection.response:int_body.SignedParts.Header_0.Name
```

and

```
SignatureProtection.response:int_body.SignedParts.Header_0.Namespace
```

There are an extensive number of combinations of settings that are available to secure your web service applications. Because of the number of attributes and configuration options from the WS-Security Version 1.0 specification, all attributes are not defined in this topic. The following sections explain the hierarchy structure for the WSSecurity policy and binding attributes:

- WSSecurity policy properties
- WSSecurity binding properties
- “setPolicyType and setBinding command examples” on page 621

WSSecurity policy properties

Use the getPolicyType command to review a properties object with the properties that are configured in your current WSSecurity policy file. Security policy schemata define the security assertions. Because the elements in the schema have hierarchical relationship, the property names for security policy also have the similar hierarchy. The hierarchical relationship between property names in the security policy is represented by a period (.) between two levels, concatenating the parent and child attributes. Examples of the properties include, but are not limited to, IncludeToken, Name, Namespace, XPath, XPathVersion. The following list describes the top-level assertion policy property names for the WSSecurity policy file:

AsymmetricBinding
You can specify zero or one binding assertion.

SymmetricBinding
You can specify zero or one binding assertion. AsymmetricBinding and SymmetricBinding cannot co-exist in a security policy file.

Wss11
You can specify zero or one Wss11 assertion.

Wss10
You can specify zero or one Wss10 assertion.

Trust10
You can specify zero or one Trust10 assertion.

SignatureProtection
You can specify zero or any number of signature protection assertions.

EncryptionProtection
You can specify zero or any number of encryption protection assertions

SupportingTokens
You can specify zero or any number of supporting token assertions.

For example, the following policy file example displays an AsymmetricBinding assertion:

```
<sp:AsymmetricBinding>
  <wsp:Policy>
    <sp:InitiatorSignatureToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
  /200512/IncludeToken/AlwaysToRecipient">
          <wsp:Policy>
            <sp:WssX509V3Token10 />
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorSignatureToken>
    <sp:RecipientSignatureToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
  /200512/IncludeToken/AlwaysToInitiator">
          <wsp:Policy>
            <sp:WssX509V3Token10 />
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientSignatureToken>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
```

```
      </sp:Layout>
    </wsp:Policy>
  </sp:AsymmetricBinding><sp:AsymmetricBinding>
    <wsp:Policy>
      <sp:InitiatorSignatureToken>
        <wsp:Policy>
          <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
    /200512/IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
              <sp:WssX509V3Token10 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:InitiatorSignatureToken>
      <sp:RecipientSignatureToken>
        <wsp:Policy>
          <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
    /200512/IncludeToken/AlwaysToInitiator">
            <wsp:Policy>
              <sp:WssX509V3Token10 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:RecipientSignatureToken>
    </wsp:Policy>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
  </sp:AsymmetricBinding>
```

The AsymmetricBinding assertion returns the following property name and value pairs. The nested `wsp:Policy` layers are not displayed in the returned properties. Additionally, some properties return the `true` value which indicates that the WSSecurity configuration includes the related XML elements. To edit these properties, set the value as `true` to include the property, or set the value as an empty string, "", to remove the property.

```
AsymmetricBinding.Layout = Strict
AsymmetricBinding.AlgorithmSuite.Basic256 = true
AsymmetricBinding.RecipientSignatureToken.X509Token_0.IncludeToken = http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToInitiator
AsymmetricBinding.InitiatorSignatureToken.X509Token_0.WssX509V3Token10 = true
AsymmetricBinding.InitiatorSignatureToken.X509Token_0.IncludeToken = http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient
AsymmetricBinding.RecipientSignatureToken.X509Token_0.WssX509V3Token10 = true
```

Additionally, the following policy file example displays a SupportingTokens assertion:

```
<sp:SupportingTokens>
        <wsp:Policy wsu:Id="request:custom_auth">
            <spe:CustomToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/
       ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                    <spe:WssCustomToken uri=http://bar.com/MyCustomToken localname="tokenv1">
                    </spe:WssCustomToken>
                </wsp:Policy>
            </spe:CustomToken>
        </wsp:Policy>
    </sp:SupportingTokens
```

The SupportingTokens assertion returns the following property name and value pairs. The nested `wsp:Policy` layers are not displayed in the returned property.

```
SupportingTokens.request:custom_auth.CustomToken_0.WssCustomToken.uri=http://bar.com
/MyCustomToken
SupportingTokens.request:custom_auth.CustomToken_0.IncludeToken=http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient
SupportingTokens.request:custom_auth.CustomToken_0.WssCustomToken.localname=tokenv1
```

Note: The CustomToken property contains a subscript zero notation (_0) because the property might be displayed multiple times from the same type of token such as the RecipientSignatureToken or InitiatorSignatureToken tokens.

Although most property names follow the hierarchical relationship format described previously, the following exceptions exist:

- The wsu:Id element

  This element uses the actual value for the ID instead of using `Id` as the attribute name. The following policy file example property:

```
<wsp:Policy wsu:Id="response:int_body">
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
</wsp:Policy>
```

  The previous wsu:Id example returns the following properties:

```
SignatureProtection.response:int_body.SignedParts.Body = true
```

- The Header element

  Because there can be multiple Header elements, the `Header_n` notation is used to represent this property. See the following policy file example:

```
 <wsp:Policy wsu:Id="request:conf_body">
     <sp:EncryptedParts>
       <sp:Body/>
       <sp:Header Name="MyElement" Namespace="http://foo.com/MyNamespace" />
     </sp:EncryptedParts>
   </wsp:Policy>
```

  The previous Header example returns the following properties:

```
EncryptionProtection.request:conf_body.EncryptedParts.Header_0.Name=MyElement
EncryptionProtection.request:conf_body.EncryptedParts.Header_0.Namespace=http://
foo.com/MyNamespace
```

- The XPath element

  The `XPath_n` notation is used to represent this property because there can be multiple XPath elements. See the following policy file example:

```
<wsp:Policy wsu:Id="request:int_body">
      <sp:SignedElements>
        <sp:XPath>SomeXPathExpression</sp:XPath>
        <sp:XPath>SomeOtherXPathExpression</sp:XPath>
      </sp:EncryptedElements>
    </wsp:Policy>
```

  The previous XPath example returns the following properties:

```
SignatureProtection.request:int_body.SignedElements.XPath_0=SomeXPathExpression
SignatureProtection.request:int_body.SignedElements.XPath_1=SomeOtherXPathExpression
```

- The X509Token element

  Use the X509Token_*n* notation to represent this property because multiple X509Token elements can exist. For an example, see the AsymmetricBinding assertion.

- The CustomToken element

  Use the CustomToken_*n* notation to represent this property because multiple CustomToken elements can exist. For an example, see the SupportingTokens assertion.

WSSecurity binding properties

Use the getBinding command to review a properties object with the properties that are configured in your current WSSecurity binding configuration. You can also use the administrative console to configure your WSSecurity bindings. Use the information center topics for configuring WSSecurity bindings with administrative console for more information.

The properties defined in this section reflect the hierarchy of the binding schema. Each part of the property name is a lowercase version of the schema type. For example, the `application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname` property follows the hierarchal format. The attributes begin with `application` or `bootstrap`. Attributes that begin with `application` represent bindings that are associated with the main WS-Security policy. Attributes that begin with `bootstrap` represent bindings that are associated with the WS-Security bootstrap policy, where the WS-Security policy uses Secure Conversation.

Some property names might have an _n notation appended to them. This notation represents a list of items. For example, multiple tokenconsumer properties exist and are listed from tokenconsumer_*0* through tokenconsumer_*n*, where the set of tokenconsumer values are:

```
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.
certpathsettings.certstoreref.reference
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.
certpathsettings.trustanchorref.reference
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
application.securityinboundbindingconfig.tokenconsumer_0.classname
application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname
application.securityinboundbindingconfig.tokenconsumer_0.name
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri
```

Additionally, some properties in the security binding file return a value of true when queried. To set these properties, set the value to true to include the property, or set the value to an empty string ("") to remove the property. For example, the time stamp, nonce, and trustAnyCertificate properties follow this pattern.

Use the setBinding command and the attributes parameter to add or remove properties to your WSSecurity binding configuration.

- To add a property, use the setBinding command to pass the property name with a non-zero length string value. To add a list item, use the _n notation to reflect a numeric value that is greater than any current numeric value for the property. For example, if the tokenconsumer_0 and tokenconsumer_1 properties exist in your configuration, specify the new tokenconsumer property as tokenconsumer_2. After adding a property, use the getBinding command to view the most recent list of configured properties.
- To remove a property, use the setBinding command to pass the property name with an empty string (""). For example, to remove all of the tokenconsumer_0 properties, specify the following property with the attributes parameter:

  ```
  application.securityinboundbindingconfig.tokenconsumer_0=""
  ```

  The previous example removes all properties that begin with the application.securityinboundbindingconfig.tokenconsumer_0 property name.

The following examples display several sets of properties to configure for your binding. This list does not include all properties to configure for the WSSecurity binding. Use this information as a reference to determine how to form specific property names.

signinginfo element

Use this property to configure signing information. For a custom binding, an unlimited number of signinginfo elements specified for the securityoutboundbindingconfig and securityinboundbindingconfig assertions can exist. In the default bindings, the system allows a maximum of two signinginfo elements for the securityoutboundbindingconfig and securityinboundbindingconfig assertions. The following example displays the format for two signinginfo elements:

```
application.securityinboundbindingconfig.signinginfo_0.signingkeyinfo_0
.reference=con_signkeyinfo
application.securityinboundbindingconfig.signinginfo_0.signingpartreference_0
.reference=request:int_body
application.securityoutboundbindingconfig.signinginfo_0.signingpartreference_0
.reference=response:int_body
application.securityoutboundbindingconfig.signinginfo_0.signingpartreference_0.timestamp=true
```

encryptioninfo element

Use this property to configure encryption information. For a custom binding, an unlimited number of encryptioninfo elements specified for the securityoutboundbindingconfig and securityinboundbindingconfig assertions can exist. In the default bindings, the system accepts a maximum of two encryptioninfo elements for the securityoutboundbindingconfig and securityinboundbindingconfig assertions. The following example displays the format for two encryptioninfo properties:

```
application.securityinboundbindingconfig.encryptioninfo_0.encryptionpartreference
.nonce=true
application.securityinboundbindingconfig.encryptioninfo_0.encryptionpartreference
.reference=request:conf_body
```

```
application.securityoutboundbindingconfig.encryptioninfo_0.encryptionpartreference
.nonce=true
application.securityoutboundbindingconfig.encryptioninfo_0.encryptionpartreference
.timestamp=true
```

tokengenerator element

In the default bindings, the `tokengenerator` elements that the `signinginfo` or `encryptioninfo` elements do not reference are considered to be authentication token generators. Each authentication token generator must have a unique `valuetype` element. The following example displays an example of a generator for an X.509 protection token:

```
application.securityoutboundbindingconfig.tokengenerator_0.name=gen_signtgen
application.securityoutboundbindingconfig.tokengenerator_0.classname=com.ibm.ws.wssecurity.wssapi.token
.impl.CommonTokenGenerator
application.securityoutboundbindingconfig.tokengenerator_0.valuetype.uri=
application.securityoutboundbindingconfig.tokengenerator_0.valuetype.localname=http://docs.oasis-open.org
/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.classname=com.ibm.websphere.wssecurity
.callbackhandler.X509GenerateCallbackHandler
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.alias=soaprequester
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.keypass={xor}PDM2OjEr
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.name=CN=SOAPRequester,
OU=TRL, O=IBM, ST=Kanagawa, C=JP
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.path=${USER_INSTALL_ROOT}
/etc/ws-security/samples/dsig-sender.ks
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.storepass={xor}PDM2OjEr
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type=JKS
application.securityoutboundbindingconfig.tokengenerator_0.jaasconfig.configname=system.wss.generate.x509
```

The following example displays a generator for a username authentication token:

```
application.securityoutboundbindingconfig.tokengenerator_1.name=gen_usernametoken
application.securityoutboundbindingconfig.tokengenerator_1.classname=com.ibm.ws.wssecurity
.wssapi.token.impl.CommonTokenGenerator
application.securityoutboundbindingconfig.tokengenerator_1.valuetype.uri=
application.securityoutboundbindingconfig.tokengenerator_1.valuetype.localname=http://docs
.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.classname=com.ibm
.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.basicAuth.userid=user1
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.basicAuth.password=myPassword
application.securityoutboundbindingconfig.tokengenerator_1.securityTokenReference.reference=request:uname_token
application.securityoutboundbindingconfig.tokengenerator_1.jaasconfig.configname=system.wss.generate.unt
```

tokenconsumer element

In the default bindings, the `tokenconsumer` elements that the `signinginfo` or `encryptioninfo` elements do not reference are authentication token consumers. Each authentication token consumer must have a unique `valuetype` element. The following example displays the format for a set of `tokenconsumer` elements:

```
application.securityinboundbindingconfig.tokenconsumer_0.name=con_unametoken
application.securityinboundbindingconfig.tokenconsumer_0.classname=com.ibm.ws.wssecurity.wssapi
.token.impl.CommonTokenConsumer
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname=http://docs.oasis-open.org
/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri=
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname=com.ibm.websphere
.wssecurity.callbackhandler.UNTConsumeCallbackHandler
application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname=system.wss.consume.unt
application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference=request:uname_token
```

actor element

Defines the actor uniform resource identifier (URI) to be included in the WSSecurity headers of a generated message, as displayed by the following example:

```
application.securityinboundbindingconfig.actor=http://myActor.com
application.securityoutboundbindingconfig.actor=http://myActor.com
```

certstorelist element

Defines certificate store configurations and signing information, as displayed by the following example:

```
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.name=DigSigCertStore
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.provider=IBMCertPath
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.x509certificates_0.path=${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
```

keyinfo element

Defines key information for signing and encryption configurations, as displayed by the following example:

```
application.securityinboundbindingconfig.keyinfo_0.classname=com.ibm.ws.wssecurity.wssapi
.CommonContentConsumer
application.securityinboundbindingconfig.keyinfo_0.name=con_signkeyinfo
application.securityinboundbindingconfig.keyinfo_0.tokenreference.reference=con_tcon
application.securityinboundbindingconfig.keyinfo_0.type=STRREF
```

trustanchor property

Defines configuration information that is used to validate the trust of the signer certificate, as displayed by the following example:

```
application.securityinboundbindingconfig.trustanchor_0.keystore.path=${USER_INSTALL_ROOT}
/etc/ws-security/samples/dsig-receiver.ks
application.securityinboundbindingconfig.trustanchor_0.keystore.storepass={xor}LDotKTot
application.securityinboundbindingconfig.trustanchor_0.keystore.type=JKS
application.securityinboundbindingconfig.trustanchor_0.name=DigSigTrustAnchor
```

timestampexpires element

Defines an expiration date for the configuration, as displayed by the following example:

```
application.securityoutboundbindingconfig.timestampexpires.expires=5
```

application.securityinboundbindingconfig.caller_X.order

Specifies the order for a caller when using wsadmin scripts, where X is the unique string that identifies the instance of the caller:

```
-attributes [[application.securityinboundbindingconfig.caller_0.order 2]]
```

setPolicyType and setBinding command examples

Use the previous reference information with the setPolicyType and setBinding commands to modify your policy and binding configuration data.

Note: The administrative console command assistance provides incorrect Jython syntax for the setPolicyType command. The XPath expression for the response message part protection of the Username WSSecurity policy set contains single quotes (') within each XPath property value, which Jython does not support. To fix the command from the administrative console command assistance, add a backslash character (\) before each single quote to escape the single quote.

The following example uses the setBinding command to set the enabled and provides properties for the myCustomSecurityPS custom policy set, which contains a ReliableMessaging policy:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSSecurity
 -attributes [[application.securityinboundbindingconfig.caller_0.order 2][inResponsewithSSL:configAlias
NodeDefaultSSLSettings]
[inResponsewithSSL:config properties_directory/ssl.client.props]
[outAsyncResponsewithSSL:configFile properties_directory/ssl.client.props]
[outAsyncResponsewithSSL:configAlias NodeDefaultSSLSetings]
[outRequestwithSSL:configFile properties_directory/ssl.client.props]
[outRequestwithSSL:configAlias NodeDefaultSSLSettings]]]')
```

The following setPolicyType command enables the WSSecurity policy and creates a signature protection assertion:

```
AdminTask.setPolicyType('-policySet myPolicySet -policyType WSSecurity -attributes
"[[enabled true][provides
Some_amount_of_security][SignatureProtection.request:app_signparts.SignedElements.XPath_0
SignatureProtectionV2]]"')
```

The following setBinding command specifies key information for a server-specific binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[[server server1][node
node01]]"
-attributes "[[application.securityinboundbindingconfig.keyinfo_0.name dec_server_keyinfo]
[application.securityinboundbindingconfig.keyinfo_0.classname
com.ibm.ws.wssecurity.wssapi.CommonContentGenerator]
[application.securityinboundbindingconfig.keyinfo_0.type STRREF]]"')
```

The following setBinding command specifies key information for an attachment-specific binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[[application PolicySet]
[attachmentId 999]]"
-attributes "[[application.securityinboundbindingconfig.keyinfo_0.name dec_app_keyinfo]
[application.securityinboundbindingconfig.keyinfo_0.classname
com.ibm.ws.wssecurity.wssapi.CommonContentGenerator]
[application.securityinboundbindingconfig.keyinfo_0.type STRREF]]" -attachmentType application
 -bindingName myBindingName')
```

The following setBinding command specifies trust anchor information for a cell-wide binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "" -attributes
"[application.securityinboundbindingconfig.trustanchor_0.name DigSigTrustAnchor2]"')
```

WSReliableMessaging policy and binding properties

Use the attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the ReliableMessaging policy and policy set binding. The WSReliableMessaging quality of service (QoS) is only available for application policy sets.

WSReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. Use WSReliableMessaging to secure and verify transactions when using web services between businesses.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

WSReliableMessaging policy properties

Configure the WSReliableMessaging policy by specifying the following properties with the setPolicyType command:

specLevel

Choose the WS-ReliableMessaging standard to use for reliable transmission of your messages. The WS-ReliableMessaging specification Version 1.1 is the default value. Use the following information to choose a specification level:

- Specify `1.0` as the value for the specLevel attribute to use the WS-ReliableMessaging specification Version 1.0, February 2005 specification level.
- Specify `1.1` as the value for the specLevel attribute to use the OASIS WS-ReliableMessaging specification Version 1.1, August 2006 specification level.

The following example code sets the specLevel property to the OASIS WS-ReliableMessaging specification Version 1.1, August 2006:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging"  -policyType
 WSReliableMessaging -attributes "[[specLevel 1.1]]"]')
```

inOrderDelivery

Specifies whether to process messages in the order that they are received. If you use the inOrderDelivery property, then inbound messages might be queued while waiting for earlier messages.

The following example code enables the inOrderDelivery property:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType WSReliableMessaging -attributes "[[inOrderDelivery true]]"]')
```

qualityOfService

Specifies the quality of the WSReliableMessaging service to use. Define one of the following three values for the qualityOfService attribute:

- unmanagedNonPersistent

  This setting tolerates network and remote system failures. The unmanagedNonPersistent quality of service is non-transactional. With this setting configured, messages are lost if a server fails. This quality of service is supported for all environments only if the environment is configured as a web service requester.

- managedNonPersistent

  This setting tolerates system, network, and remote system failures. However, the message state is discarded when the messaging engine restarts. The managedNonPersistent quality of service is non-transactional. This setting prevents message loss if a server fails. However, messages are lost if the messaging engine fails. Managed and thin client applications cannot use this quality of service.

- managedPersistent

  This setting tolerates system, network, and remote system failures. With this setting, messages are processed within transactions, persisted at the web service requester and provider. Messages can be recovered if a server fails. Messages that are not successfully transmitted at the time of failure continue when the messaging engine or application restarts. Managed and thin client applications cannot use this quality of service.

The following example sets the qualityOfService property as unmanaged nonpersistent:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType
 WSReliableMessaging -attributes "[[qualityOfService unmanagedNonPersistent]]"]')
```

The following example uses the setPolicyType command to set a value for each policy property:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType
 WSReliableMessaging -attributes "[[specLevel 1.1][inOrderDelivery true][qualityOfService
 unmanagedNonPersistent]]"]')
```

WSReliableMessaging binding configuration attributes

If you set the qualityOfService policy property to managedNonPersistent or managedPersistent, configure the WSReliableMessaging binding by specifying values for the following properties with the setBinding command:

busName

The name of the service integration bus that contains the messaging engine to use for the managedNonPersistent or managedPersistent Quality of Service options.

The following example sets the busName property as *myBus*:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[busName myBus]]"]')
```

messagingEngineName

The name of the messaging engine to use for the managedNonPersistent or managedPersistent quality of service options.

The following example sets the messagingEngineName property as *messagingEngine001*:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[messageEngineName messageEngine001]]"]')
```

The following code example demonstrates how to use the setBinding command to set values for each binding attribute:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[busName myBus][messageEngineName messageEngine001]]"]')
```

WSAddressing policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the WSAddressing policy and policy set binding. Application and system policy sets use the WSAddressing policy and binding.

WSAddressing is an interoperability standard for addressing Web services and providing addressing information in messages. For more information, see the W3C Candidate Recommendation (CR) versions of the WS-Addressing core and SOAP specifications.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

WSAddressing policy properties

Configure the WSAddressing policy by specifying the following properties with the setPolicyType command:

usingAddressing
Specifies whether a WS-Addressing SOAP header is included on messages. Use one of the following values:

required
WS-Addressing is mandatory. Servers return an error if they receive a message that does not contain a WS-Addressing header. Clients always include WS-Addressing headers in SOAP messages.

optional
WS-Addressing is not mandatory. Servers do not generate an error if they receive a message that does not contain a WS-Addressing header. Clients might not include WS-Addressing headers in SOAP messages, for example, if WS-Policy is enabled and the server does not specify that WS-Addressing is mandatory.

wsaMode
Specifies the messaging style that this policy set supports. Use one of the following values:

WSA_SYNC
Response messages must be targeted at the WS-Addressing anonymous URI.

WSA_ASYNC
Response messages must not be targeted at the WS-Addressing anonymous URI.

WSA_BOTH
The targeting of response messages is not restricted.

The following example uses the setPolicyType command to set WS-Addressing to mandatory, and the messaging style to synchronous, for the policy set myPolicySet:.

```
AdminTask.setPolicyType('[-policySet "myPolicySet" -policyType WSAddressing
-attributes "[[usingaddressing required][wsaMode WSA_SYNC]]"]')
```

WSAddressing binding properties

Configure the WSAddressing policy by specifying the following property with the setBinding command:

preventWLM

Specifies whether to prevent workload management for references to endpoints that were created by the application programming interface (API) in a cluster environment. Messages that target Endpoint References (EPRs) within a cluster environment are workload managed by default.

Preventing workload management routes messages that target EPRs to the node or server on which the EPR was created. You might disable workload management if the endpoint maintains the in-memory state, which has not been replicated across other nodes or servers within the cluster.

For example, the following command prevents workload management for a cell-wide general binding, from the WSAddressing policy.

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSAddressing -attributes "[preventWLM true]"]')
```

SSLTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the SSLTransport policy and policy set binding. Application and system policy sets can use the SSLTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured

general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

SSLTransport policy properties

Use the SSLTransport policy to ensure message security.

Configure the SSLTransport policy by specifying the following properties with the setPolicyType command:

outRequestSSLenabled
Specifies whether to enable the SSL security transport for outbound service requests.

outAsyncResponseSSLenabled
Specifies whether to enable the SSL security transport for asynchronous service responses.

inResponseSSLenabled
Specifies whether to enable the SSL security transport for inbound service responses.

The following setPolicyType command example sets values for all SSLTransport policy properties:

```
AdminTask.setPolicyType('[-policySet "WSHTTPS default" -policyType SSLTransport
-attributes "[[inReponseSSLenabled yes][outAsyncResponseSSLenabled yes][outRequestSSLenabled
 yes]]"]')
```

SSLTransport binding properties

Use the SSLTransport policy type to ensure message security.

Configure the SSLTransport binding by specifying the following properties using the setBinding command:

outRequestwithSSL:configFile
outRequestwithSSL:configAlias
If you enable SSL outbound service requests, then these two attributes define the specific SSL security transport binding and location. The default value for the outRequestwithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the outRequestwithSSL:configAlias attribute is `NodeDefaultSSLSettings`.

outAsyncResponsewithSSL:configFile
outAsyncResponsewithSSL:configAlias
If you enable SSL asynchronous service responses, then these two attributes define the specific SSL security transport binding and location. The default value for the outAsyncRequestwithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the outAsyncRequestwithSSL:configAlias attribute is `NodeDefaultSSLSettings`.

inResponsewithSSL:configFile
inResponsewithSSL:configAlias
If you enable SSL inbound service responses, then these two attributes define the specific SSL security transport binding and location. The default value for the inResponsewithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the inResponsewithSSL:configAlias property is `NodeDefaultSSLSettings`.

The following setBinding command example sets values for all SSLTransport binding attributes:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 SSLTransport -attributes "[[inResponsewithSSL:configAlias NodeDefaultSSLSettings] [inResponsewithSSL:config
 properties_directory/ssl.client.props][outAsyncResponsewithSSL:configFile properties_directory/ssl.client.props]
[outAsyncResponsewithSSL:configAlias NodeDefaultSSLSetings][outRequestwithSSL:configFile
 properties_directory/ssl.client.props][outRequestwithSSL:configAlias NodeDefaultSSLSettings]]"]')
```

HTTPTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the HTTPTransport policy and policy set binding. Application and system policy sets can use the HTTPTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

The following sections explain the policy and binding properties to configure:

- HTTPTransport policy properties

- HTTPTransport binding properties

HTTPTransport policy properties

The HTTPTransport policy set can be used for HTTPS, basic authorization, compression, and binary encoding transport methods.

Configure the HTTPTransport policy by specifying the following attributes with the **setPolicyType** command:

protocolVersion
Specifies the version of HTTP to use. The valid version values are `HTTP/1.1` and `HTTP/1.0`.

maintainSession
Specifies whether the HTTP session is enabled when a message is sent. The valid values are `yes` or `no`.

chunkTransferEnc
Specifies whether to enable chunked transfer encoding. The valid values are `yes` or `no`.

sendExpectHeader
Specifies whether to send an expect 100-request header. The valid values are `yes` or `no`.

compressRequest:name
Specifies whether to compress the request. The valid values are `gzip`, `x-gzip`, `deflate`, or `none`.

compressResponse:name
Specifies whether to compress the response. The valid values are `gzip`, `x-gzip`, `deflate`, or `none`.

acceptRedirectionURL
Specifies whether to accept URL redirection automatically. The valid values are `yes` or `no`.

messageResendOnce
Specifies if a message can be sent more than once. The valid values are `yes` or `no`.

connectTimeout
Specifies the amount of time, in seconds, before a connection times out when sending a message. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 180 seconds. No maximum value is set for this property.

writeTimeout
Specifies the amount of time, in seconds, before the write time out occurs. Specify an integer value. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 300 seconds. No maximum value is set for this property.

readTimeout
Specifies the amount of time, in seconds, before the read time out occurs. Specify an integer value. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 300 seconds. No maximum value is set for this property.

persistConnection
Specifies whether to use a persistent connection when sending messages. Valid values are `yes` or `no`.

The following setPolicyType example command sets values for each HTTPTransport binding property:

```
AdminTask.setPolicyType('[-policySet "WSHTTPS custom" -policyType HTTPTransport
-attributes "[[protocolVersion HTTP/1.1]
[sessionEnable yes][chunkTransferEnc yes][sendExpectHeader yes]
```

```
[compressRequest:name gzip][compressResponse:name
 gzip][acceptRedirectionURL yes][messageResendOnce no][connectTimeout
300][writeTimeout 300]
[readTimeout 300][persistConnection yes]]"]')
```

HTTPTransport binding properties

Configure the HTTPTransport binding by specifying the following attributes with the setBinding command:

outAsyncResponseBasicAuth:userid
Specifies the user name for basic authentication of outbound asynchronous responses.

outAsyncResponseBasicAuth:password
Specifies the password for basic authentication of outbound asynchronous responses.

outAsyncResponseProxy:userid
Specifies the user name for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:password
Specifies the password for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:port
Specifies the port number for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:host
Specifies the host name for the outbound asynchronous service responses proxy.

outRequestBasicAuth:userid
Specifies the user name or basic authentication of outbound service requests.

outRequestBasicAuth:password
Specifies the password for basic authentication of outbound service requests.

outRequestProxy:userid
Specifies the user name for the outbound service request proxy.

outRequestProxy:password
Specifies the password for the outbound service request proxy.

outRequestProxy:port
Specifies the port number for the outbound service request proxy.

outRequestProxy:host
Specifies the host name for the outbound service request proxy.

The following **setBinding** example command sets values for each HTTPTransport binding property:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName generalCellWideBind1 -policyType
HTTPTransport
-attributes "[[outAsyncResponseBasicAuth:userid myID][outAsyncResponseBasicAuth:password
myPW][outAsyncResponseProxy:host hostname]
[outAsyncResponseProxy:port 9060][outAsyncResponseProxy:userid myID]
[outAsyncResponseProxy:password myPW]
[outRequestBasicAuth:userid myID][outRequestBasicAuth:password myPW]
[outRequestProxy:userid myID]
[outRequestProxy:password myPW][outRequestProxy:port 9061][outRequestProxy:host
 hostname]]"]')
```

JMSTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the JMSTransport policy and policy set binding. Application policy sets can use the JMSTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.

- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

The following sections explain the policy and binding properties to configure:

- JMSTransport policy properties
- JMSTransport binding properties

JMSTransport policy properties

Use the JMSTransport policy set to configure JMS transport for applications that use the Java Messaging Service (JMS) to exchange request and response messages.

Configure the JMSTransport policy by specifying the following attributes with the setPolicyType command:

requestTimeout

Specifies the request timeout value. The request timeout value is the amount of time, in seconds, that the client waits for a response after sending the request to the server. The default value is `300` seconds. If you specify an integer value of zero or less, the system sets the requestTimeout property to the default value of `300` seconds. No maximum value exists for this property.

allowTransactionalAsyncMessaging
Specifies whether a client uses transactions in one-way or asynchronous two-way requests. The default value for this property is `false`. Set the value of this property to `true` to enable transactional messaging. When enabled, the client runtime exchanges SOAP request and response messages with the server over the JMS transport in a transactional manner if the client operates under a transaction.

The client transaction is used to send the SOAP request message to the destination queue or topic, and the server receives the request message only after the client commits the transaction. Similarly, the server receives the request message under the control of a container-managed transaction and sends the reply message, if applicable, back to the client using that same transaction. Then, the client receives the reply message after the server transaction is committed.

The following setPolicyType example command sets values for each JMSTransport binding property:

```
AdminTask.setPolicyType('[-policySet "JMS custom" -policyType JMSTransport
 -attributes "[[requestTimeout 300][allowTransactionalAsynMessaging false]]"]')
```

JMSTransport binding properties

Configure the JMSTransport binding by specifying the following attributes with the setBinding command:

outRequestBasicAuth:userid
Specifies the user name or basic authentication of outbound service requests.

outRequestBasicAuth:password
Specifies the password for basic authentication of outbound service requests.

The following **setBinding** example command sets values for each HTTPTransport binding property:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName generalCellWideBind1
 -policyType JMSTransport -attributes "[[outRequestBasicAuth:userid myID] [outRequestBasicAuth:password myPW]]"]')
```

CustomProperties policy and binding properties

Use the -attributes parameter for the setBinding command to specify additional configuration information for the CustomProperties policy set binding. Application and system policy sets can use the CustomProperties policy and binding.

Note: This product supports using CustomProperties policy and binding to set generic properties that are not supported in other policy types. The additional properties are set in the binding. You must only update existing properties, such as the enabled attribute, in the policy. The CustomProperties policy provides an alternative way to set a binding property instead of using the JAX-WS programming model to set the property on the BindingProvider object. The CustomProperties binding is only supported for service clients.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **-attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **-attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **-attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **-attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might

not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the SystemOut.log file for any error and warning messages that indicate that the input for the **-attributes** parameter contains one or multiple properties that are not valid.

Note: This topic references one or more of the application server log files. Beginning in WebSphere Application Server Version 8.0 you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using SystemOut.log , SystemErr.log, trace.log, and activity.log files or native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The following setBinding example command demonstrates how you can use the CustomProperties binding to set a value for the WSADDRESSING_DESTINATION_EPR endpoint reference binding property:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName generalCellWideBind1 -attachmentType client
 -policyType CustomProperties -attributes "[[WSADDRESSING_DESTINATION_EPR addressValue]]"]')
```

Note: In a mixed cell environment, the following limitations apply to attachments to policy sets containing CustomProperties policy:

- You must not create attachments to policy sets containing CustomProperties policy for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. The CustomProperties policy is only supported on WebSphere Application Server V8 and later.
- An application that contains an attachment to a policy set containing CustomProperties policy must not be deployed on an application server that is prior to WebSphere Application Server Version 8.
- If an application that is deployed in a cluster environment contains an attachment to a policy set containing CustomProperties policy, you must not add a member application server that is prior to WebSphere Application Server Version 8 to the cluster.

SecureConversation command group for the AdminTask object (Deprecated)

Use this topic as a reference for the commands for the SecureConversation group of the AdminTask object. Use these commands with your administrative scripts to query, update, and remove secure conversation client cache configuration data.

Note: The commands in the SecureConversation command group are deprecated. Use the commands in the WSSCacheManagement command group to manage WS-Security distributed cache configurations.

Use the following commands in the SecureConversation group to manage your custom and non-custom secure conversation client cache configurations:

querySCClientCacheConfiguration command

The querySCClientCacheConfiguration command lists all non-custom client cache configuration data for WS-SecureConversation.

Target object

None

Required parameters

None.

Optional parameters

None.

Return value

This command returns a list of all non-custom client cache configuration data.

Batch mode example usage

- Using Jython:

```
print AdminTask.querySCClientCacheConfiguration()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySCClientCacheConfiguration('-interactive')
```

querySCClientCacheCustomConfiguration command

The querySCClientCacheCustomConfiguration command lists all custom client cache configuration data for WS-SecureConversation.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a list of all custom client cache configuration data.

Batch mode example usage

- Using Jython:

```
print AdminTask.querySCClientCacheCustomConfiguration()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySCClientCacheCustomConfiguration('-interactive')
```

updateSCClientCacheConfiguration command

The updateSCClientCacheConfiguration command sets the cache cushion time in minutes and enables or disables distributed cache.

Target object

None.

Required parameters

None.

Optional parameters

-distributedCache
Specifies whether distributed cache is enabled or disabled. If you set the -distributedCache parameter to `true` when you run the updateSCClientCacheConfiguration command, the system enables distributed cache for the WS-Security runtime. (Boolean, optional)

-minutesInCacheAfterTimeout
Specifies the amount of time, in minutes, that the token remains in the cache after it expires. The token is renewable for this amount of time. (Integer, optional)

-renewIntervalBeforeTimeoutMinutes
Specifies the amount of time, in minutes, that a renew request is allowed before the token expires. (Integer, optional)

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSCClientCacheConfiguration('-minutesInCacheAfterTimeout 100 -distributedCache
 true')
```

- Using Jython list:

```
AdminTask.updateSCClientCacheConfiguration(['-minutesInCacheAfterTimeout', '100', '-distributedCache',
'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSCClientCacheConfiguration('-interactive')
```

updateSCClientCacheCustomConfiguration command

The updateSCClientCacheCustomConfiguration command updates custom properties for the secure conversation client cache configuration.

Target object

None.

Required parameters

None.

Optional parameters

-customProperties
The custom properties for the secure conversation client cache configuration. (Properties, optional)

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSCClientCacheCustomConfiguration('[-customProperties "[ [property2 value2]
[property1 value1] ]"]')
```

- Using Jython list:

```
AdminTask.updateSCClientCacheCustomConfiguration(['-customProperties', '[ [property2 value2]
[property1 value1] ]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSCClientCacheCustomConfiguration('-interactive')
```

deleteSCClientCacheConfigurationCustomProperties command

The deleteSCClientCacheConfigurationCustomProperties command removes specific properties from a custom secure conversation client cache configuration.

Target object

None.

Required parameters

-propertyNames
The names of the properties to delete. (String, required).

Optional parameters

None.

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteSCClientCacheConfigurationCustomProperties('[-propertyNames [property1,property2]]')
```

- Using Jython list:

```
AdminTask.deleteSCClientCacheConfigurationCustomProperties(['-propertyNames', '[property1,property2]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSCClientCacheConfigurationCustomProperties('-interactive')
```

WSSCacheManagement command group for the AdminTask object

Use this topic as a reference for the commands for the WSSCacheManagement group of the AdminTask object. Use these commands with your administrative scripts to query, update, and remove distributed cache configuration data.

Use the following commands in the WSSCacheManagement group to manage your custom and non-custom distributed cache configurations:

- "deleteWSSDistributedCacheConfigCustomProperties command"
- "queryWSSDistributedCacheConfig command"
- "queryWSSDistributedCacheCustomConfig command" on page 638
- "updateWSSDistributedCacheConfig command" on page 639
- "updateWSSDistributedCacheCustomConfig command" on page 640

deleteWSSDistributedCacheConfigCustomProperties command

The deleteWSSDistributedCacheConfigCustomProperties command removes WS-Security distributed cache custom properties.

Target object

None

Required parameters

-propertyNames
Specifies the names of the custom properties to delete from the distributed cache configuration. (String[])

Optional parameters

None.

Return value

This command returns a message that indicates the success or failure of the command.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('[-propertyNames
 [prop1,prop2,prop3]]')
```

- Using Jython list:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties([',-'propertyNames',
'[prop1,prop2,prop3]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('-interactive')
```

queryWSSDistributedCacheConfig command

The queryWSSDistributedCacheConfig command lists the WS-Security distributed cache configuration non-custom properties.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a properties object that contains the configuration properties and values for the distributed cache configuration. The following table displays the configuration properties that the command returns:

Table 108. Command properties. Use the command to list distributed cache configuration properties.

Property	Description
tokenRecovery	Specifies whether token recovery is enabled or disabled. If the tokenRecovery property is set to `true`, the Datasource property specifies the shared data source that is assigned to the distributed cache.
distributedCache	Specifies whether distributed caching is enabled or disabled.
Datasource	Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled.
renewIntervalBeforeTimeoutMinutes	Specifies the amount of time, in minutes, that the client waits before it attempts to renew the token.
synchronousClusterUpdate	Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled.
minutesInCacheAfterTimeout	Specifies the amount of time that the token remains in the cache after the token times out.

Batch mode example usage

- Using Jython:

```
print AdminTask.queryWSSDistributedCacheConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheConfig('-interactive')
```

queryWSSDistributedCacheCustomConfig command

The queryWSSDistributedCacheCustomConfig command lists the WS-Security distributed cache configuration custom properties.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a properties object that contains the name and value pairs that correspond to each custom property.

Batch mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheCustomConfig('-interactive')
```

updateWSSDistributedCacheConfig command

The updateWSSDistributedCacheConfig command updates the WS-Security distributed cache configuration non-custom properties.

Target object

None.

Required parameters

None.

Optional parameters

-renewIntervalBeforeTimeoutMinutes
Specifies the amount of time, in minutes, that a renew request is allowed before the token expires. (Integer)

-minutesInCacheAfterTimeout
Specifies the amount of time, in minutes, that the token remains in the cache after it expires. The token is renewable for this amount of time. (Integer)

-distributedCache
Specifies whether distributed cache is enabled or disabled. (Boolean)

-synchronousClusterUpdate
Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled. Specify `false` to disable synchronous cluster updating. (Boolean)

-tokenRecovery
Specifies whether token recovery is enabled or disabled. If you set the tokenRecovery property to `true`, specify the shared data source to assign to the distributed cache with the -Datasource parameter. (Boolean)

-Datasource
Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled. (String)

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateWSSDistributedCacheConfig('[-customProperties "[ [property2 value2]
 [property1 value1] ]"]')
```

- Using Jython list:

```
AdminTask.updateWSSDistributedCacheConfig(['-customProperties', '[ [property2 value2]
[property1 value1] ]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateWSSDistributedCacheConfig('-interactive')
```

updateWSSDistributedCacheCustomConfig command

The updateWSSDistributedCacheCustomConfig command updates the WS-Security distributed cache configuration custom properties.

Target object

None.

Required parameters

-customProperties
Specifies the name and value of each custom property to add or update in the WS-Security distributed cache configuration. (java.util.Properties)

Optional parameters

None.

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateWSSDistributedCacheCustomConfig('[-customProperties [[property1 value1]
[property2 value2]]]')
```

- Using Jython list:

```
AdminTask.updateWSSDistributedCacheCustomConfig(['-customProperties', '[[property1 value1]
[property2 value2]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateWSSDistributedCacheCustomConfig('-interactive')
```

PolicySetManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Use the following commands to manage policy set configurations:

- “listPolicySets ” on page 641
- “getPolicySet ” on page 642

- “createPolicySet ” on page 643
- “copyPolicySet ” on page 643
- “deletePolicySet ” on page 644
- “updatePolicySet ” on page 645
- “validatePolicySet ” on page 647
- “exportPolicySet ” on page 647
- “importPolicySet ” on page 648

Use the following commands to manage policy settings:
- “addPolicyType ” on page 645
- “deletePolicyType ” on page 646
- “listPolicyTypes ” on page 648
- “getPolicyType ” on page 650
- “setPolicyType ” on page 650
- “getPolicyTypeAttribute ” on page 651
- “setPolicyTypeAttribute ” on page 652

Use the following commands to manage policy set attachments:
- “getPolicySetAttachments ” on page 653
- “createPolicySetAttachment ” on page 654
- “updatePolicySetAttachment ” on page 656
- “addToPolicySetAttachment ” on page 657
- “removeFromPolicySetAttachment ” on page 659
- “deletePolicySetAttachment ” on page 660
- “listAttachmentsForPolicySet ” on page 662
- “listAssetsAttachedToPolicySet” on page 662
- “deleteAttachmentsForPolicySet ” on page 663
- “transferAttachmentsForPolicySet ” on page 664
- “listSupportedPolicySets” on page 665

Use the following commands to manage policy set bindings:
- “getBinding ” on page 666
- “setBinding ” on page 667
- “getDefaultBindings” on page 669
- “getRequiredBindingVersion ” on page 670
- “setDefaultBindings” on page 671
- “exportBinding ” on page 671
- “importBinding ” on page 672
- “copyBinding ” on page 673
- “upgradeBindings” on page 674

listPolicySets

The listPolicySets command returns a list of all existing policy sets. If administrative security is enabled, each user role can use this command.

Target object

None.

Optional parameters

-policySetType
Specifies the type of policy set. Specify `application` to display application policy sets. Specify `system` to display system policy sets for trust service or WS-MetadataExchange attachments. Specify `system/trust` to display the policy sets for the trust service. Specify `default` to display the default policy sets. The default value for this parameter is `application`. (String, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of all existing policy sets. Each entry in the list is the name of a policy set.

Batch mode example usage

- Using Jython string:

```
AdminTask.listPolicySets('[-policySetType system/trust]')
```

- Using Jython list:

```
AdminTask.listPolicySets(['-policySetType', 'system/trust'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listPolicySets('-interactive')
```

getPolicySet

The getPolicySet command returns general attributes, such as description and default indicator, for the specified policy set. If administrative security is enabled, each user role can use this command.

Target object

None.

Required parameters

-policySet
Specifies the policy set name. For a list of all policy set names, use the listPolicySets command. (String, required)

Optional parameters

-isDefaultPolicySet
Specifies whether to display a default policy set. The default value is `false`. (Boolean, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of attributes for the specified policy set name.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.getPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicySet('-interactive')
```

createPolicySet

The createPolicySet command creates a new policy set. Policies are not created with the policy set. The default indicator is set to `false`.

If administrative security is enabled, you must use the Administrator role to create policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set. (String, required)

Optional parameters

-description
Adds a description for the policy set. (String, required)

-policySetType
Specifies the type of policy set. When the value is `application`, the command creates application policy sets. When the value is `system`, the command creates a policy set that you can use for trust service or WS-MetadataExchange attachments. When the value is `system/trust`, the command creates a policy set for the trust service. The default value for this parameter is `application`. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.createPolicySet('[-policySet myCustomPS -description [my new custom policy
 set] -policySetType system/trust]')
```

- Using Jython list:

```
AdminTask.createPolicySet(['-policySet', 'myCustomPS', '-description',  '[my new custom
 policy set]', '-policySetType', 'system/trust'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createPolicySet('-interactive')
```

copyPolicySet

The copyPolicySet command creates a copy of an existing policy set. By default, the policy set attachments are transferred to the new policy set.

If administrative security is enabled, you must use the Administrator role to copy policy sets.

Target object

None.

Required parameters

-sourcePolicySet
Specifies the name of the existing policy set to copy. (String, required)

-newPolicySet
Specifies the name of the new policy set you are creating. (String, required)

-newDescription
Specifies a description for the new policy set. (String, required)

Optional parameters

-transferAttachments
If this parameter is set to `true`, all attachments transfer from the source policy set to the new policy set. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.copyPolicySet('[-sourcePolicySet SecureConversation -newPolicySet
 CustomSecureConversation -newDescription [my new copied policy set] -transferAttachments true]')
```

- Using Jython list:

```
AdminTask.copyPolicySet(['-sourcePolicySet', 'SecureConversation', '-newPolicySet',
 'CustomSecureConversation', '-newDescription', '[my new copied policy set]',  '-transferAttachments',
 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.copyPolicySet('-interactive')
```

deletePolicySet

The deletePolicySet command deletes the specified policy set. If attachments exist for the policy set, the command returns a failure message.

If administrative security is enabled, you must use the Administrator role to delete policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to delete. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicySet('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.deletePolicySet(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicySet('-interactive')
```

updatePolicySet

The updatePolicySet command enables you to input an attribute list to update the policy set. You can use this command to update all attributes for the policy set, or a subset of attributes.

If administrative security is enabled, you must use the Administrator role to update policy set configurations.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-attributes
Specifies a properties object that contains the attributes to update for the specified policy set. (Properties, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updatePolicySet('-policySet policySet1 -attributes [[type application]
 [description [my policy set description]]]')
```

- Using Jython list:

```
AdminTask.updatePolicySet(['-policySet', 'policySet1', '-attributes', '[[type
 application] [description [my policy set description]]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updatePolicySet('-interactive')
```

addPolicyType

The addPolicyType command adds a policy with default values for the specified policy set. You must indicate whether to enable or disable the added policy.

If administrative security is enabled, you must use the Administrator role to add policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-policyType
Specifies the name of the policy to add to the policy set. (String, required)

-enabled
If this parameter is set to `true`, new policy is enabled in the policy set. If this parameter is set to `false`, the configuration is contained within the policy set but the configuration does not have an effect on the system. (Boolean, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addPolicyType('[-policySet customPolicySet -policyType WSTransaction
 -enabled true]')
```

- Using Jython list:

```
AdminTask.addPolicyType(['-policySet', 'customPolicySet', '-policyType',
 'WSTransaction', '-enabled', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addPolicyType('-interactive')
```

deletePolicyType

The deletePolicyType command deletes a policy from a policy set.

If administrative security is enabled, you must use the Administrator role to remove policies from your configuration.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-policyType
Specifies the name of the policy to remove from the policy set. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicyType('[-policySet customPolicySet -policyType WSTransaction]')
```

- Using Jython list:

```
AdminTask.deletePolicyType(['-policySet', 'customPolicySet', '-policyType',
 'WSTransaction'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicyType('-interactive')
```

validatePolicySet

The validatePolicySet command validates the policy set configuration.

If administrative security is enabled, you must use the Administrator role to validate policy sets.

Target object

None.

Required parameters

-policySet
Specifies the policy set to update. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.validatePolicySet('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.validatePolicySet(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.validatePolicySet('-interactive')
```

exportPolicySet

The exportPolicySet command exports a policy set as an archive that can be copied onto a client environment.

If administrative security is enabled, you must use the Administrator role to export policy sets.

Target object

None.

Required parameters

-policySet
Specifies the policy set to export. (String, required)

-pathName
Specifies the path name of the archive file to create. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportPolicySet('[-policySet customSecureConversation -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip]')
```

- Using Jython list:

```
AdminTask.exportPolicySet(['-policySet', 'customSecureConversation;, '-pathName', '
 C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportPolicySet('-interactive')
```

importPolicySet

The importPolicySet command imports a policy set from a compressed archive file or from a selection of default policy sets onto the server environment.

If administrative security is enabled, you must use the Administrator role to import policy sets.

Target object

None.

Optional parameters

-importFile
Specifies the path name of the archive file to import. (String, optional)

-defaultPolicySet
Specifies the name of the default policy set to import. (String, optional)

-policySet
Specifies the name to assign to the new policy set. If you do not specify this parameter, the system uses the original name of the policy set. (String, optional)

-verifyPolicySetType
Specifies that the policy set type to import matches a specific type. Specify `system` or `system/trust` to verify that the policy set to import is a type of system policy set, including trust service policy sets. Specify `application` to verify that the policy set is an application policy set. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.importPolicySet('[-importFile C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip]')
```

- Using Jython list:

```
AdminTask.importPolicySet(['-importFile', 'C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importPolicySet('-interactive')
```

listPolicyTypes

The listPolicyTypes command returns a list of the names of the policies configured on your system. The input parameters allow you to list each policy type configured in the system, the policy types configured in a policy set, or the policy types in a binding.

If administrative security is enabled, each administrative role can list policy types.

Target object

None.

Optional parameters

-policySet
Specifies the name of the policy set to query for policies. If the policy set is not specified, the command lists all policies defined in your configuration. (String, optional)

-bindingLocation
Specifies the location of the binding. This value is cell-wide default binding, server-specific default binding, or attachment-specific binding. Specify the bindingLocation parameter as a properties object following these guidelines:

- For cell-wide default binding, use a null or empty properties.
- For server-specific default binding, specify the node and server names in the properties. The property names are `node` and `server`. Server-specific default bindings are deprecated.
- For attachment-specific binding, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For system/trust bindings, set the `systemType` property as `trustService`.
- For trust client bindings, specify the `systemType` property as `trustClient`. In addition, specify the attachment ID. If the bindings are for a specific application, also specify the `application` property.
- For WSNClient binding, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`.

(Properties, optional)

-attachmentType
Specifies whether the attachment type is an application binding, client binding, trust service binding, trust client binding, or WS-Notification client binding. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-bindingName
Specifies a specific general binding. If you specify this parameter, the system displays policy types in the specific binding. (String, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of policy types.

Batch mode example usage

- Using Jython string:

```
AdminTask.listPolicyTypes('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.listPolicyTypes(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listPolicyTypes('[-interactive]')
```

getPolicyType

The getPolicyType command returns the attributes for a specified policy.

If administrative security is enabled, each administrative role can query attributes for policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to query. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

Optional parameters

-attributes
Specifies the specific attributes to display. If this parameter is not used, the command returns all attributes for the specified policy. (String[], optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a properties object containing the policy attributes.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicyType('[-policySet customSecureConversation -policyType
 SecureConversation]')
```

- Using Jython list:

```
AdminTask.getPolicyType(['-policySet', 'customSecureConversation', '-policyType', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicyType (['-interactive'])
```

setPolicyType

The setPolicyType command updates the attributes of a specified policy.

Note: The administrative console command assistance provides incorrect Jython syntax for the setPolicyType command. The XPath expression for the response message part protection of the Username WSSecurity policy set contains single quotes (') within each XPath property value, which Jython does not support. To fix the command from the administrative console command assistance, add a backslash character (\) before each single quote to escape the single quote.

Also, if you are using a Jython script to update the attributes, the brackets should not be included if you want to get a list of elements and not a list of strings.

If administrative security is enabled, you must use the Administrator role to configure policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributes
Specifies the specific attributes to be updated. The properties could include all of the policy attributes or a subset of attributes. (Properties, required)

Optional parameters

-replace
Indicates whether the new attributes provided from the command replace the existing policy attributes. For policies with complex data, you can remove optional parts of the configuration when necessary. Use this parameter to get all attributes, perform edits, and replace the binding configuration with the edited data. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.setPolicyType('[-policySet customSecureConversation -policyType
 SecureConversation -attributes [[type application] [description [my new description]]]]')
```

- Using Jython list:

```
AdminTask.setPolicyType(['-policySet', 'customSecureConversation', '-policyType', 'SecureConversation',
 '-attributes',
 '[[type application] [description [my new description]]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setPolicyType('-interactive')
```

getPolicyTypeAttribute

The getPolicyTypeAttribute command returns the value for the specified policy attribute.

If administrative security is enabled, each administrative role can query policy type attribute values.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributeName
Specifies the name of the attribute of interest. (String, required)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Optional parameters

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a string that contains the value of the specified attribute.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicyTypeAttribute('[-policySet customSecureConversation  -policyType
 SecureConversation -attributeName type]')
```

- Using Jython list:

```
AdminTask.getPolicyTypeAttribute(['-policySet', 'customSecureConversation',  '-policyType',
 'SecureConversation', '-attributeName', 'type'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicyTypeAttribute('-interactive')
```

setPolicyTypeAttribute

The setPolicyTypeAttribute command sets the value for the specified policy attribute.

If administrative security is enabled, you must use the Administrator role to configure policy attributes.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributeName
Specifies the name of the attribute of interest. (String, required)

-attributeValue
Specifies the value of the attribute of interest. (String, required)

Return value

If the attribute is successfully added to the policy, the command returns the `true` string value.

Batch mode example usage

- Using Jython string:

```
AdminTask.setPolicyTypeAttribute('[-policySet customPolicySet  -policyType
 WSReliableMessaging -attributeName specLevel  -attributeValue 1.0]')
```

- Using Jython list:

```
AdminTask.setPolicyTypeAttribute(['-policySet', 'customPolicySet',  '-policyType',
 'WSReliableMessaging', '-attributeName', 'specLevel',  '-attributeValue', '1.0'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setPolicyTypeAttribute('-interactive')
```

getPolicySetAttachments

The getPolicySetAttachments command lists the properties for all policy set attachments configured in a specified application.

If administrative security is enabled, each administrative role can query for policy set attachments.

Target object

None.

Optional parameters

-applicationName
Specifies the name of the application to query for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required to query for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-expandResources
Provides expanded information that details the attachment properties for each resource. If you set this parameter to the name of the service, only the resources for that web service are returned. If you specify an asterisk (*) character, expanded information for all your Web services is returned. This parameter is valid if the value for the -attachmentType parameter is set to `provider` or `client`. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

-serviceRef
Specifies the name of the service reference for which the attachments are returned. If specified, only

attachments for the service reference are returned. This parameter is only valid when the expandResources parameter value is the name of your service and when the attachmentType parameter is set to client. (String, optional)

Return value

The command returns a list of properties for each attachment in the application, including the policy set name, attachment ID, and resource list. If you specify the expandResources parameter, the command returns the resource, attachmentId, policySet, binding, and directAttachment properties. If a resource is not attached to a policy set, then the system only displays the resource property. The binding property only exists if the attachment contains a custom binding.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicySetAttachments('[-attachmentType provider -attachmentProperties
"[systemType trustService]"]')
```

- Using Jython list:

```
AdminTask.getPolicySetAttachments(['-attachmentType', 'provider', '-attachmentProperties',
 '[systemType trustService]'])
```

The following examples return policy set attachments information for the specified service reference, `myServiceRef`. The examples return detailed resource information for the logical endpoints or operations for each service reference because the -expandResource parameter is specified.

- Using Jython string:

```
AdminTask.getPolicySetAttachments('[-attachmentType client -applicationName application1 -expandResources
 {http://www.ibm.com}myService -serviceRef myServiceRef]')
```

- Using Jython list:

```
AdminTask.getPolicySetAttachments(['-attachmentType', 'client', '-applicationName', 'application1',
'-expandResources', '{http://www.ibm.com}myService', -serviceRef', 'myServiceRef'])
```

Interactive mode example usage

- Using Jython list:

```
AdminTask.getPolicySetAttachments('-interactive')
```

createPolicySetAttachment

The createPolicySetAttachment command creates a new policy set attachment for an application.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 109. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can create policy set attachments for application resources only.
Operator	The Operator role cannot create policy set attachments.
Monitor	The Monitor role cannot create policy set attachments.

Target object

None.

Required parameters

-resources
Specifies the name of the application resources to attach to the policy set. (String[], required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to `true`, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

-inheritFromService
Specifies whether the resources for the service reference inherit the policy set attachments of the associated service. Use this parameter for service reference attachments only. The default value for this parameter is `true`. (Boolean, optional)

-policySet
Specifies the name of the policy set to attach. This parameter is required unless the resource specifies a service reference and the inheritFromService parameter is specified. If the policySet parameter is not specified and the inheritFromService parameter is`false`, all attachments for the service reference are removed, and the service reference does not have a policy set attachment. If the policySet parameter is not specified and the inheritFromService parameter is `true`, all attachments for the service reference are removed, and the service reference inherits the policy of the service. (String, optional)

Return value

The command returns a string with the ID of the new attachment.

Batch mode example usage

- Using Jython string:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources "WebService:/" -applicationName
WebService -attachmentType provider]')
```

- Using Jython list:

```
AdminTask.createPolicySetAttachment(['-policySet', 'policyset1', '-resources', '"WebService:/"',
'-applicationName', 'WebService',
 '-attachmentType', 'provider'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createPolicySetAttachment('-interactive')
```

Note: In a mixed cell environment, you must not create service reference attachments or resource attachments that are specified in name-value pair format for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. Service reference attachments are only supported on WebSphere Application Server V8 and later.

In a mixed cell environment, you must not create attachments to policy sets containing CustomProperties policy for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. The CustomProperties policy is only supported on WebSphere Application Server V8 and later.

updatePolicySetAttachment

The updatePolicySetAttachment command updates the resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 110. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure policy set attachments. If you have access to a specific resource only, you can configure policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to configure policy set attachments. If you have access to a specific resource only, you can configure policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can configure policy set attachments for application resources only.
Operator	The Operator role cannot configure policy set attachments.
Monitor	The Monitor role cannot configure policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to update. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. A resource for a service reference cannot be included in the same attachment as a resource for a service. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to `true`, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updatePolicySetAttachment('[-attachmentId 123 -resources "WebService:/"]')
```

- Using Jython list:

```
AdminTask.updatePolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updatePolicySetAttachment ('-interactive')
```

Note: In a mixed cell environment, you must not create service reference attachments or resource attachments that are specified in name-value pair format for applications that are deployed on an application server that is prior to Version 8.0. Service reference attachments are only supported on Version 8.0 and later.

In a mixed cell environment, you must not create attachments to policy sets containing CustomProperties policy for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. The CustomProperties policy is only supported on WebSphere Application Server V8 and later.

addToPolicySetAttachment

The addToPolicySetAttachment command adds additional resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 111. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to add resources to policy set attachments. If you have access to a specific resource only, you can add resources to policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to add resources to policy set attachments. If you have access to a specific resource only, you can add resources to policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can add resources to policy set attachments for application resources only.
Operator	The Operator role cannot add resources to policy set attachments.
Monitor	The Monitor role cannot add resources to policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to update. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. A resource for a service reference cannot be included in the same attachment as a resource for a service. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to `true`, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addToPolicySetAttachment('[-attachmentId 123 -resources
 "WebService:/webapp1.war:{http://www.ibm.com}myService"]')
```

- Using Jython list:

```
AdminTask.addToPolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/webapp1.war:{http://www.ibm.com}myService"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addToPolicySetAttachment('-interactive')
```

Note: In a mixed cell environment, you must not create service reference attachments or resource attachments that are specified in name-value pair format for applications that are deployed on an application server that is prior to Version 8.0. Service reference attachments are only supported on Version 8.0 and later.

In a mixed cell environment, you must not create attachments to policy sets containing CustomProperties policy for applications that are deployed on an application server that is prior to WebSphere Application Server Version 8. The CustomProperties policy is only supported on WebSphere Application Server V8 and later.

removeFromPolicySetAttachment

The removeFromPolicySetAttachment command removes resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 112. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to remove resources from policy set attachments. If you have access to a specific resource only, you can remove resources for which you have access.
Configurator	The Configurator role must have cell-wide access to remove resources from policy set attachments. If you have access to a specific resource only, you can remove the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can remove resources from policy set attachments for application resources only.
Operator	The Operator role cannot remove resources from policy set attachments.
Monitor	The Monitor role cannot remove resources from policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to remove. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeFromPolicySetAttachment('[-attachmentId 123 -resources
 "WebService:/webapp1.war:{http://www.ibm.com}myService"]')
```

- Using Jython list:

```
AdminTask.removeFromPolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/webapp1.war:{http://www.ibm.com}myService"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeFromPolicySetAttachment('-interactive')
```

deletePolicySetAttachment

The deletePolicySetAttachment command removes a policy set attachment from an application.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 113. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.

Table 113. Administrative roles (continued). This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Configurator	The Configurator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can delete policy set attachments for application resources only.
Operator	The Operator role cannot delete policy set attachments.
Monitor	The Monitor role cannot delete policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to delete. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. If a trust client attachment is specified, the -attachmentProperties parameter contains a `systemType` property with a value of `trustClient`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicySetAttachment('[-attachmentId 123]')
```

- Using Jython list:

```
AdminTask.deletePolicySetAttachment(['-attachmentId', '123'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicySetAttachment('-interactive')
```

listAssetsAttachedToPolicySet

The listAssetsAttachedToPolicySet command lists the applications or WS-Notification service clients to which a specific policy set is attached.

If administrative security is enabled, each administrative role can list applications that are attached to policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachments. The value for this parameter must be `provider`, `client`, `WSNClient`, `WSMex`, `cuProvider`, `cuClient`, `binding` or `all`. The default value is `all`. (String, optional)

Return value

The command returns a list of properties that describe each asset. Each properties object contains the `assetType` property, which specifies the type of asset.

Batch mode example usage

- Using Jython string:

```
AdminTask.listAssetsAttachedToPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.listAssetsAttachedToPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAssetsAttachedToPolicySet('-interactive')
```

listAttachmentsForPolicySet

The listAttachmentsForPolicySet command lists the applications to which a specific policy set is attached.

If administrative security is enabled, each administrative role can query for policy set attachments.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachments. The value for this parameter must be `application`, `client`, or `system/trust`. The default value is `application`. (String, optional)

Return value

The command returns a list of application names.

Batch mode example usage

- Using Jython string:

```
AdminTask.listAttachmentsForPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.listAttachmentsForPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAttachmentsForPolicySet('-interactive')
```

deleteAttachmentsForPolicySet

The deleteAttachmentsForPolicySet command removes all attachments for a specific policy set.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 114. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can delete policy set attachments for application resources only.
Operator	The Operator role cannot delete policy set attachments.
Monitor	The Monitor role cannot delete policy set attachments.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set from which to remove the attachments. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest. The command only deletes attachments for the application of interest if you specify this parameter. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. You can specify values for the `bus` and `WSNService` properties. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAttachmentsForPolicySet('[-policySet customSecureConversation
 -applicationName newApp1]')
```

- Using Jython list:

```
AdminTask.deleteAttachmentsForPolicySet(['-policySet', 'customSecureConversation',
 '-applicationName', 'newApp1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAttachmentsForPolicySet('-interactive')
```

transferAttachmentsForPolicySet

The transferAttachmentsForPolicySet command transfers all attachments from one policy set to another policy set.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 115. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to transfer policy set attachments. If you have access to a specific resource only, you can transfer policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to transfer policy set attachments. If you have access to a specific resource only, you can transfer policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can transfer policy set attachments for application resources only.
Operator	The Operator role cannot transfer policy set attachments.
Monitor	The Monitor role cannot transfer policy set attachments.

Target object

None.

Required parameters

-sourcePolicySet
Specifies the source policy set from which to copy attachments. (String, required)

-destinationPolicySet
Specifies the name of the policy set to which the attachments are copied. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest. The command only transfers attachments for the application of interest if you specify this parameter. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. You can specify values for the `bus` and `WSNService` properties. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet SecureConversation
 -destinationPolicySet customSecureConversation -applicationName newApp1]')
```

- Using Jython list:

```
AdminTask.transferAttachmentsForPolicySet(['-sourcePolicySet', 'SecureConversation',
 '-destinationPolicySet', 'customSecureConversation', '-applicationName', 'newApp1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.transferAttachmentsForPolicySet('-interactive')
```

listSupportedPolicySets

The listSupportedPolicySets command returns a list of supported policy sets to attach to your web services resources. If administrative security is enabled, each user role can use this command.

Target object

None.

Required parameters

-assetProps
Specifies the name of the asset of interest. Specify the name of the application as the value for the **application** property. (Properties, required)

Supported property and value pairs for the parameter are:

Property	**Value**
application	Application name, such as *myApplication*
WS-Notification service client: • bus • WSNService	Service client names: • Bus name, such as: *bus1* • Service name, such as *service1*
Trust service resource: • systemType	Service resoure name: • trustService
SCA business-level application resource: • blaName • cuName	Resource names: • business-level application name, such as *myBLA* • composition unit name, such as *compositionUnit1*

Return value

The command returns a list of supported policy sets. Each entry in the list is the name of a policy set.

Batch mode example usage

- Using Jython string:

```
AdminTask.listSupportedPolicySets ('[-assetProps [[application myApplication]]]')
```

- Using Jython list:

```
AdminTask.listSupportedPolicySets (['-assetProps', '[[bus bus1] [WSNService service1]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSupportedPolicySets('-interactive')
```

getBinding

The getBinding command returns the binding configuration for a specified policy and scope. You can use the getBinding command to return a list of available custom bindings, which includes bindings that are and are not referenced by attachments.

If administrative security is enabled, each administrative role can query for binding configuration information.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Target object

None.

Required parameters

-policyType
Specifies the policy of interest. (String, required)

-bindingLocation
Specifies the location of the binding. (Properties, required)

Specify the bindingLocation parameter as a properties object following these guidelines:

- For WebSphere Application Server Version 7.0 and later server default bindings, specify a null or empty properties. Use the bindingName parameter to identify the binding location.
- For attachment-specific bindings, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For WSNClient bindings, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`. If you specify an asterisk character (*) as the attachment ID, then the command returns the list of binding names that corresponds to the attachment type of interest.

- For system/trust bindings, specify the `systemType` property as `trustService`.

Optional parameters

-attachmentType
Specifies the type of policy set attachment. Use this parameter to distinguish between types of attachment custom bindings. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attributes
Specifies the names of the attributes to return. If this parameter is not specified, the command returns all attributes. (String, optional)

-bindingName
Specifies the binding name of interest. Specify this parameter to display a general cell-level binding or a custom attachment binding. (String, optional)

Return value

The command returns a properties object that contains the requested configuration attributes for the policy binding.

Batch mode example usage

- Using Jython string:

  The following example returns a list of application bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType provider
 -bindingLocation [[application application_name] [attachmentId *]]]')
```

  The following example returns a list of client bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType client
 -bindingLocation [[application application_name] [attachmentId *]]]')
```

  The following example returns a list of system bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType provider
 -bindingLocation [[systemType trustService] [application application_name] [attachmentId *]]]')
```

- Using Jython list:

```
AdminTask.getBinding(['-policyType', 'WSAddressing', '-bindingLocation', '""'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getBinding('-interactive')
```

setBinding

The setBinding command updates the binding configuration for a specified policy. Use this command to add a server-specific binding, update an attachment to use a custom binding, edit binding attributes, or to remove a binding configuration.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Table 116. Administrative roles. This table describes the administrative roles and associated authorization when administrative security is enabled.

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure custom bindings for the resource for which you have access. The Administrator role is the only role that can modify binding configurations.
Configurator	The Configurator role must have cell-wide access to assign and unassign bindings. If you have access to a specific resource only, you can assign and unassign bindings for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings for application resources only.
Operator	The Operator role cannot configure bindings.
Monitor	The Monitor role cannot configure bindings.

Note: In WebSphere Application Server Version 7.0 and later, the security model was enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0 and later, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 and later Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Target object

None.

Required parameters

-bindingLocation
Specifies the location of the binding. (Properties, required)

Specify the bindingLocation parameter as a properties object following these guidelines:

- For WebSphere Application Server Version 7.0 and later server default bindings, specify a null or empty properties. Use the bindingName parameter to identify the binding location.
- For attachment-specific, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For WSNClient bindings, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`. If you specify an asterisk character (*) as the attachment ID, then the command returns the list of binding names that corresponds to the attachment type of interest.
- For system/trust bindings, set the `systemType` property as `trustService`.

-policyType
Specifies the policy of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachment. Use this parameter to distinguish between types of attachment custom bindings. (String, optional)

Note: Even though you can specify the `application` value for the -attachmentType parameter, use the `provider` value in place of the `application` value because the attachments are used for more than just applications, such as system attachments for trust service. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attributes
Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update. If the **attributes** parameter is not specified, the command only updates the binding location used by the specified attachment. (Properties, optional)

-bindingName
Specifies the name for the binding. Specify this parameter to assign a new name to an attachment binding or cell-level binding. A name is generated if it is not specified. (String, optional)

-domainName
Specifies the domain name for the binding. This parameter is required when using the command to create and scope a binding to a specific domain other than the administrative security domain. The default value is `global`. (String, optional)

-replace
Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is false. (Boolean, optional)

-remove
Specifies whether to remove a server-specific default binding or to remove a custom binding from an attachment. You cannot remove cell-level default binding. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.setBinding('[-policyType WSAddressing -bindingLocation [[application myApplication]
[attachmentId 123]] -attributes "[preventWLM false]"
 -attachmentType provider]')
```

- Using Jython list:

```
AdminTask.setBinding(['-policyType', 'WSAddressing', '-bindingLocation', '[[application myApplication]
 [attachmentId 123]]', '-attributes', '[preventWLM
 false]', '-attachmentType', 'provider'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setBinding('-interactive')
```

getDefaultBindings

The getDefaultBindings command displays the provider and client default bindings if the bindings are set. If the command does not return output, then the system default binding is the current default.

If administrative security is enabled, each administrative role can query for default bindings.

Target object

None.

Optional parameters

-bindingLocation
Specifies the location of the binding. Specify the bindingLocation parameter as a properties object with values for the `node` and `server` properties. (Properties, optional)

-domainName
Specifies the domain name for the binding of interest. This parameter is required if the domain of interest is not in the global security domain and you did not specify the bindingLocation parameter. The bindingLocation and domainName parameters are mutually exclusive. The default value is `global`. (String, optional)

Return value

The command returns a properties object that contains the names of the provider and client default bindings, if the bindings are set.

Batch mode example usage

- Using Jython string:

```
AdminTask.getDefaultBindings('[-bindingLocation [[node myNode] [server myServer]]]')
```

- Using Jython list:

```
AdminTask.getDefaultBindings(['-bindingLocation', '[[node myNode] [server myServer]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getDefaultBindings('-interactive')
```

getRequiredBindingVersion

The getRequiredBindingVersion command displays the version number of the binding for a specific application.

Target object

None.

Optional parameters

-assetProps
Specifies the name of the application of interest. (Properties, optional)

Return value

The command returns the binding version number as a number, such as `7.0.0.0` or `6.1.0.0`.

Batch mode example usage

- Using Jython string:

```
AdminTask.getRequiredBindingVersion('[-assetProps [[application myApplication]]]')
```

- Using Jython list:

```
AdminTask.getRequiredBindingVersion(['-assetProps', '[[application myApplication]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getRequiredBindingVersion('-interactive')
```

setDefaultBindings

The setDefaultBindings command to set a binding as the default binding.

If administrative security is enabled, you must use the Administrator role with cell-wide access to configure bindings. If you use the Administrator role and do not have cell-wide access, you can only configure bindings on resources for which you have access.

Target object

None.

Required parameters

-defaultBindings
Specifies the names of the default bindings for the provider, client, or both. (Properties, required)

Optional parameters

-bindingLocation
Specifies the location of the binding. Specify the bindingLocation parameter as a properties object with values for the `node` and `server` properties. (Properties, optional)

-domainName
Specifies the domain name for the binding of interest. This parameter is required if the domain of interest is not in the global security domain and you did not specify the bindingLocation parameter. The bindingLocation and domainName parameters are mutually exclusive. The default value is `global`. (String, optional)

Return value

The command returns a value of `true` if the command successfully sets the default binding.

Batch mode example usage

- Using Jython string:

```
AdminTask.setDefaultBindings('[-defaultBindings [[provider myDefaultProviderBinding]
[client myDefaultClientBinding]] -bindingLocation [[node myNode] [server myServer]]]')
```

- Using Jython list:

```
AdminTask.setDefaultBindings(['-defaultBindings', '[[provider myDefaultProviderBinding'
 [client myDefaultClientBinding]]', '-bindingLocation', '[[node
 myNode] [server myServer]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setDefaultBindings('-interactive')
```

exportBinding

The exportBinding command export a general, cell-level binding to an archive file. You can copy this file to a client environment or import the archive to a server environment.

If administrative security is enabled, you must use the Administrator role with cell-wide access to export bindings.

Target object

None.

Required parameters

-bindingName
Specifies the name of the binding to assign as the default binding. If you do not specify this parameter, the system specifies the system default as the default binding. (String, required)

-pathName
Specifies the file path for the archive file to create. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportBinding('[-bindingName myDefaultBinding -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/Bindings/]')
```

- Using Jython list:

```
AdminTask.exportBinding(['-bindingName', 'myDefaultBinding', '-pathName',
 'C:/IBM/WebSphere/AppServer/PolicySets/Bindings/'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportBinding('-interactive')
```

importBinding

The importBinding command imports a general, cell-level binding from a compressed archive file to a server environment.

If administrative security is enabled, you must use the Administrator role with cell-wide access to import bindings.

Target object

None.

Required parameters

-pathName
Specifies the file path for the archive file to import. (String, required)

Optional parameters

-bindingName
Specifies the name of the binding to assign as the imported binding. If you do not specify this parameter, the system specifies the binding name in the archive file. (String, optional)

-domainName
Specifies a new name of the domain of the binding to import. If you do not specify this parameter, the command uses the domain specified in the archive file. (String, optional)

-verifyBindingType
Verifies that the type of binding to import matches a specific binding type. Specify `provider` to verify that the binding to import is a provider binding, or specify `client` to verify that it is a client binding. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.importBinding('[-bindingName myDefaultBinding -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/Bindings/myBinding.ear]')
```

- Using Jython list:

```
AdminTask.importBinding(['-bindingName', 'myDefaultBinding', '-pathName',
 'C:/IBM/WebSphere/AppServer/PolicySets/Bindings/myBinding.ear'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importBinding('-interactive')
```

copyBinding

The copyBinding command creates a new general, cell-level binding from an existing binding.

If administrative security is enabled, you must use the Administrator role with cell-wide access to copy bindings.

Target object

None.

Required parameters

-sourceBinding
Specifies the name of the existing binding that the system uses to create the new binding. (String, required)

-newBinding
Specifies the name of the binding to create. (String, required)

Optional parameters

-newDescription
Specifies the description text for the new binding. (String, optional)

-domainName
Specifies the domain name for the binding. This parameter is only required if you scope the binding to a domain other than the domain of the source binding. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.copyBinding('[-sourceBinding mySourceBinding -newBinding mySourceCopyBinding]')
```

- Using Jython list:

```
AdminTask.copyBinding(['-sourceBinding', 'mySourceBinding', '-newBinding',
 'mySourceCopyBinding'])
```

Interactive mode example usage

- Using Jython list:

```
AdminTask.copyBinding('-interactive')
```

upgradeBindings

The upgradeBindings command upgrades application bindings for a specific asset to the latest version.

If administrative security is enabled, you must use the Administrator role with cell-wide access to import bindings.

Target object

None.

Required parameters

-assetProps
Specifies the name of the asset of interest. Specify the name of the application as the value for the **application** property. (Properties, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.upgradeBindings('[-assetProps [[application myApplication]]]')
```

- Using Jython list:

```
AdminTask.upgradeBindings(['-assetProps', '[[application myApplication]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.upgradeBindings('-interactive')
```

WS-Policy commands for the AdminTask object

You can manage WS-Policy settings for web service resources by using wsadmin scripting. You can view or manage how a service provider shares its policies, and how a service client obtains and applies the policies of a service provider.

To run these commands, use the AdminTask object of the wsadmin scripting client. Each command acts on multiple objects in one operation. The commands are provided to allow you to make the most commonly-required types of update in a consistent manner, where modifying the underlying objects directly would be error-prone.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

These commands are valid only when they are used with WebSphere Application Server Version 7 and later application servers. Do not use them with earlier versions.

The commands to manage WS-Policy settings for web service resources are part of the PolicySetManagement command group for the AdminTask object.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using these commands, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

The commands that are listed as subtopics are available to manage WS-Policy settings in the PolicySetManagement group of the AdminTask object.

The following commands are available to manage WS-Policy settings in the PolicySetManagement group of the AdminTask object:

- getProviderDynamicPolicyControl command
- setProviderDynamicPolicyControl command
- getProviderPolicySharingInfo command
- setProviderPolicySharingInfo command
- getClientDynamicPolicyControl command
- setClientDynamicPolicyControl command

getProviderPolicySharingInfo command:

Use the getProviderPolicySharingInfo command to find out whether an application or service that is a web service provider can share its policy configuration, and list the properties that apply to sharing that configuration.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the getProviderPolicySharingInfo command to find out how a web services application, or a service in a Web services application, shares its policy configuration with clients, service registries, or services that support the WS-Policy specification. The policy configuration is shared in WS-PolicyAttachments format.

The command returns properties that show whether the policy configuration of the resource can be shared with clients through a WS-MetadataExchange request or through Web Services Description Language (WSDL) that is obtained by a `?WSDL` HTTP Get request.

Target object

None.

Required parameters

`-applicationName`
The name of the application for which you want to find out how it shares its policy configuration. The application must be a service provider. (String)

Optional parameters

`-resource`
The name of the resource for which you want to find out how it shares its policy configuration. If you specify this parameter, only the properties for that resource are returned. To retrieve information for the application, specify `WebService:/`. Alternatively, you can specify a service, endpoint or operation. However, policy sets are attached only at the application or service level, so the properties returned for an endpoint or operation are the settings that are inherited from the service. (String)

Return value

Returns a list of properties that include the resource name and that show whether the policy configuration of the resource can be shared. The following properties can be returned:

`wsMexPolicySetName`
The name of the policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the **sharePolicyMethods** property is wsMex and a policy set to provide message-level security was specified.

`wsMexPolicySetBinding`
The name of the binding that is applied when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the **sharePolicyMethods** property is wsMex and a binding to provide message-level security was specified.

`resource`
The resource that you specified.

`directSetting`
How the properties apply to the resource. Valid values for this property are:

`true`
The properties apply directly to the resource.

false
The properties are inherited from the parent application or service.

sharePolicyMethods
How the policy configuration of the resource can be shared. Valid values for this property are:

httpGet
The resource shares its policy configuration through an HTTP Get request.

wsMex
The resource shares its policy configuration through a WS-MetadataExchange request.

Example

The following command displays the policy sharing configuration properties for the EchoService service in the WSSampleServices application. The provider is configured to share its policy through an HTTP Get request, and a WS-MetadataExchange request with message-level security. Message-level security for the WS-MetadataExchange request is provided by using the SystemWSSecurityDefault policy set and the "Provider sample" general binding.

```
AdminTask.getProviderPolicySharingInfo(['-applicationName', 'WSSampleServices',
'-resource', 'WebService:/SampleServicesSei.war:{http://example_path/}EchoService'])
.
.
[ [wsMexPolicySetName SystemWSSecurityDefault] [wsMexPolicySetBinding [Provider sample]]
[resource WebService:/SampleServicesSei.war:{http://example_path/}EchoService/]
[directSetting true] [sharePolicyMethods [httpGet wsMex]] ]
```

setProviderPolicySharingInfo command:

Use the setProviderPolicySharingInfo command to set how an application or service that is a web service provider can share its policy configuration with other clients, service registries, or services that support the WS-Policy specification. You can set or remove this information about how a provider policy is shared.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic "Configure Qshell to run WebSphere Application Server scripts".

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the setProviderPolicySharingInfo command to set how an application, or a service in an application, shares its policy configuration with clients, service registries, or services that support the WS-Policy specification. The policy configuration is shared in WS-PolicyAttachments format.

The policy configuration of the resource can be shared with clients through a WS-MetadataExchange request, through Web Services Description Language (WSDL) exported by a ?WSDL HTTP Get request, or through both methods.

Target object

None.

Required parameters

-applicationName
The name of the application for which you want to set how the provider policy is shared. (String)

-resource
The name of the resource for which you want to set how the provider policy is shared. For all resources in an application, specify `WebService:/`. For a service in an application, specify `WebService:/module:{namespace}service_name`. Endpoints or operations inherit the settings of the parent application or service. (String)

Optional parameters

-sharePolicyMethods
Specifies how the policy configuration of the resource can be shared. (String array)

Enter either or both of the following values:

httpGet
The resource can share its policy configuration through WSDL that is obtained by a `?WSDL` HTTP Get request.

wsMex
The resource can share its policy configuration through a WS-MetadataExchange request.

-wsMexProperties
Specifies that message-level security is required for WS-MetadataExchange requests and specifies the settings that provide the message-level security. (Properties)

Enter the following values, following each value with the setting that you require for that value:

wsMexPolicySetName
The name of the system policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. Specify a system policy set that contains only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault.

wsMexPolicySetBinding
The name of the general binding for the policy set attachment when the resource shares its policy configuration through a WS-MetadataExchange request. Specify a general binding that is scoped to the global domain, or scoped to the security domain of this service. If you do not specify this property, the default binding is used.

This parameter is valid only when you specify wsMex for the **sharePolicyMethods** parameter.

-remove
Specifies whether the information about how the provider policy is shared is removed from the resource. (Boolean)

This parameter takes the following values:

true The information about how the provider policy is shared is removed from the resource.

false This value is the default. The information about how the provider policy is shared is not removed from the resource.

Examples

The following example removes the information about how the provider policy is shared from the WSSampleServices application:

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/ -remove true]')
```

The following example enables policy sharing, using WSDL exported by a `?WSDL` HTTP Get request, for the EchoService service in the WSSampleServices application:

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/WSSampleServicesSei.war:{http://example_path/}EchoService
-sharePolicyMethods [httpGet ]]')
```

The following example enables policy sharing, using a WS-MetadataExchange request with message-level security, for the WSSampleServices application. Message level security is provided by the SystemWSSecurityDefault policy set and the “Provider sample” general binding.

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/ -sharePolicyMethods [wsMex ]
-wsMexProperties [ [wsMexPolicySetName [SystemWSSecurityDefault]]
[wsMexPolicySetBinding [Provider sample]] ]]')
```

getClientDynamicPolicyControl command:

Use the getClientDynamicPolicyControl command to find out whether an application that is a web service client obtains the policy configuration of a web service provider, and to list the properties that apply to obtaining that configuration.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the getClientDynamicPolicyControl command to find out how an application, or a service in an application, obtains the policy configuration of a service provider. The client can obtain the policy configuration of the provider through a Web Services Metadata Exchange (WS-MetadataExchange) request or through an HTTP GET request.

Target object

None.

Required parameters

-applicationName
The name of the application for which you want to find out how it obtains the policy configuration of a service provider. The application must be a service client. (String)

Optional parameters

-resource
The name of the resource for which you want to find out how it obtains the policy configuration of a service provider. If you specify this parameter, only the properties for that resource are returned. To retrieve information for the application, specify `WebService:/`. Alternatively, you can specify a service, a service reference, an endpoint, or an operation. (String)

For further details, see the topic Configuring the client policy to use a service provider policy by using wsadmin scripting.

Return value

Returns a list of properties that include the resource name and that show how it obtains the policy configuration of a service provider. The following properties can be returned:

httpGetTargetURI
The target URL of the HTTP GET request. This property is returned if the value of the **acquireProviderPolicyMethod** property is httpGet.

httpGetPolicySetName
The system policy set that contains the HTTP and SSL transport policy to use for the HTTP GET request. This property is returned if the **httpGetTargetURI** property has a value.

httpGetPolicySetBinding
The general binding that contains the HTTP and SSL transport bindings for the HTTP GET request. This property is returned if the **httpGetTargetURI** property has a value.

wsMexPolicySetName
The name of the policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the **acquireProviderPolicyMethod** property is wsMex and a policy set to provide message-level security was specified.

wsMexPolicySetBinding
The name of the binding that is used when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the **acquireProviderPolicyMethod** property is wsMex and a binding to provide message-level security was specified.

acquireProviderPolicyMethod
How the policy configuration of the provider can be obtained. Valid values for this property are:

wsMex
The resource can obtain the policy configuration of a service provider through a WS-MetadataExchange request.

httpGet
The resource can obtain the policy configuration of a service provider through an HTTP GET request.

resource
The resource that you specified.

directSetting
How the properties apply to the resource. Valid values for this property are:

true
The properties apply directly to the resource.

false
The properties are inherited from the parent application or service.

Examples

The following example displays the properties that control how the EchoService service of the WSPolicyClient application obtains the policy configuration of a service provider. The client is configured to retrieve the provider policy through a WS-MetadataExchange request with message-level security, by using the SystemWSSecurityDefault policy set and the “Client sample” general binding.

```
AdminTask.getClientDynamicPolicyControl(['-applicationName', 'WSPolicyClient',
'-resource', 'WebService:/WSPClient.war:{http://example_path/}EchoService'])
.
.
[ [wsMexPolicySetName SystemWSSecurityDefault] [wsMexPolicySetBinding [Client sample]]
[acquireProviderPolicyMethod [wsMex]]
[resource WebService:/WSPClient.war:{http://example_path/}EchoService/]
[directSetting true] ]
```

The following example displays the properties that control how the EchoService service of the WSPolicyClient application obtains the policy configuration of a service provider when the client is configured to retrieve the provider policy through an HTTP GET request.

```
AdminTask.getClientDynamicPolicyControl(['-applicationName', 'WSPolicyClient',
'-resource', 'WebService:/WSPClient.war:{http://example_path/}EchoService'])
.
.
[ [httpGetTargetURI http://example_path/EchoService?wsdl]
[acquireProviderPolicyMethod [httpGet]]
[resource WebService:/WSPClient.war:{http://example_path/}EchoService/]
[directSetting true] ]
```

setClientDynamicPolicyControl command:

Use the setClientDynamicPolicyControl command to set how an application that is a web services client obtains the policy configuration of a web services provider. You can set, refresh, or remove this information about how a provider policy is obtained.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the setClientDynamicPolicyControl command to set how a client obtains the policy configuration of a service provider.

The client can obtain the policy configuration of the provider through a Web Services Metadata Exchange (WS-MetadataExchange) request or through an HTTP GET request. The service provider must publish its policy in WS-PolicyAttachment format in its Web Services Description Language (WSDL) and the client must be able to support those provider policies.

At run time, the client uses the information to establish a policy configuration that is acceptable to both the client and the service provider.

Target object

An application or service that is a web services client.

Required parameters

-applicationName
The name of the application for which you want to obtain the policy configuration of the provider. (String)

-resource
The name of the resource for which you want to obtain the policy configuration of the provider. For all resources in an application, specify `WebService:/`. Alternatively, you can specify a service or service reference.

See the Configuring the client policy to use a service provider policy by using wsadmin scripting topic for further details.

Optional parameters

-acquireProviderPolicyMethod
Specifies how the policy configuration of the provider can be obtained. (String)

Enter one of the following values:

httpGet
Obtain the policy configuration of the provider by using an HTTP GET request.

By default, the HTTP GET request is targeted at the URL for each service endpoint followed by `?WSDL`. If you specify a service for the **resource** parameter, and you want to specify a different location for the policy configuration of the provider, you can use the **httpGetProperties** parameter to change the target of the request.

By default, the HTTP GET request uses the same HTTP and SSL transport policies as the application request. If you use the **httpGetProperties** parameter to change the target of the request, and you want to specify different HTTP and SSL transport policies for the request, you can specify the system policy set and general binding that contains the HTTP and SSL transport policies you require.

wsMex Obtain the policy configuration of the provider by using a WS-MetadataExchange request.

By default, the WS-MetadataExchange request inherits the policy set and binding configuration from the application. You can specify the system policy set and general binding that contains the WS-Security policies you require.

-wsMexProperties

Specifies that message-level security is required for WS-MetadataExchange requests and specifies the settings that provide the message-level security. (Properties)

Enter the following values, following each value with the setting that you require for that value:

wsMexPolicySetName
The name of the system policy set that specifies message-level security for a WS-MetadataExchange request. Specify a system policy set that contains only WS-Security policies, only WS-Addressing policies, or both.

wsMexPolicySetBinding
The name of the general binding for the policy set attachment for a WS-MetadataExchange request. Specify a general binding that is scoped to the global domain, or scoped to the security domain of this service. If you do not specify this property, the default binding is used.

This value is valid only when you specify the **wsMexPolicySetName** value.

This parameter is valid only when you specify `wsMex` for the **acquireProviderPolicyMethod** parameter.

-httpGetProperties

Specifies the target for acquiring provider policy by using an HTTP GET request if the provider policy is at a different location from the target endpoint. Optionally specifies the system policy set and general binding that contains the HTTP and SSL transport policies you require. (Properties)

Enter the following values, followed by the setting that you require for each value:

httpGetTargetURI
The URL for the location of the provider policy. For example, the location might refer to policy held in a registry.

httpGetPolicySetName
The name of the system policy set that contains the HTTP and SSL transport policy to use for the HTTP GET request. If the specified system policy set contains policy types other than HTTP and SSL transport, these additional policy types are ignored.

This value is valid only when you specify the **httpGetTargetURI** value.

httpGetPolicySetBinding
The name of the general binding that contains the HTTP and SSL transport bindings for the HTTP GET request. If you do not specify this property, the default binding is used.

This value is valid only when you specify the **httpGetPolicySetName** value.

This parameter is valid only when you specify `httpGet` for the **acquireProviderPolicyMethod** parameter and the resource is a service. Do not use this parameter if the resource is an application.

-remove

Specifies whether to remove the information about how the client obtains the policy configuration of the provider. (Boolean)

This parameter takes the following values:

true Information about how the client obtains the policy configuration of the provider is removed.

false This value is the default. Information about how the client obtains the policy configuration of the provider is not removed.

Examples

The following example removes the information about how the client obtains the policy configuration of the provider from the EchoService service of the WSPolicyClient client application.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-remove true]')
```

The following example configures the EchoService service of the WSPolicyClient client application to obtain the policy configuration of the provider by using an HTTP GET request.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-acquireProviderPolicyMethod [httpGet ]
-httpGetProperties [httpGetTargetURI http://example_path]]')
```

The following example configures the EchoService service of the WSPolicyClient client application to obtain the policy configuration of the provider by using an HTTP GET request. The request uses the HTTP and SSL transport policies contained in the SystemWSSecurityDefault policy set and the “Client sample” general binding.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-acquireProviderPolicyMethod [httpGet ]
-httpGetProperties [ [httpGetTargetURI http://example_path]
[httpGetPolicySetName SystemWSSecurityDefault]
[httpGetPolicySetBinding [Client sample]] ]]')
```

The following example configures the EchoService service of the WSPolicyClient client application to obtain the policy configuration of the provider through a WS-MetadataExchange request with message-level security, by using the SystemWSSecurityDefault policy set and the “Client sample” general binding.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-acquireProviderPolicyMethod [wsMex ]
-wsMexProperties [ [wsMexPolicySetName [SystemWSSecurityDefault]]
[wsMexPolicySetBinding [Client sample]] ]]')
```

Configuring secure sessions between clients and services using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting language, to edit trust service configurations. Use the STSManagement command group for the AdminTask object to specify details related to secure sessions between clients and target services.

About this task

The trust service uses the secure messaging mechanisms of the Web Services Trust (WS-Trust) specification to define additional extensions for issuing, exchanging, and validating security tokens. Use the STSManagement command group for the AdminTask object to configure the trust service using the wsadmin tool. Complete any of the following tasks using the STSManagement commands:

Procedure

- Manage token provider configurations.

Use the wsadmin tool to manage token providers. Customize token providers by defining properties such as token type schema URI, handler factory, cache cushion time, class name, and token timeout. You can also allow or restrict the use of post-dated tokens, distributed cache, and renewable tokens after timeout.

- Query existing token provider configurations.

  Use the wsadmin tool to query the existing trust service token provider configuration.
- Manage endpoint token assignments.

  Use the wsadmin tool to assign, unassign, and modify endpoint token assignments.
- Refresh your configuration changes.

  Use the wsadmin tool to force the trust service to reload the token provider configuration during run time. Complete this action to use new configuration changes before you restart the application server.

What to do next

Use the information center topics for managing token providers using the STSManagement group of commands and the AdminTask object.

Querying the trust service using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to query the trust service for existing configuration settings. Use the commands in this topic to view current trust service configurations before adding, removing, or editing token provider and endpoint configurations.

About this task

Query your current token provider configurations or endpoint configurations using the STSManagement group of commands. Use the following Jython syntax command examples when writing automation scripts to retrieve configuration attributes and set the output to a variable. Pass the newly set variable to administrative commands in the STSManagement group to automate the editing of token provider and endpoint configurations.

Procedure

- Use the following command examples to query the trust service for token provider configurations.
  - Determine the local name of the default token provider and set it to the myDefaultTokenType variable.

    The following command sets the *myDefaultTokenType* variable to the local name string for the default token provider:

    ```
    myDefaultTokenType = AdminTask.querySTSDefaultTokenType()
    print myDefaultTokenType
    ```
  - List the local names of each configured token provider.

    The following command sets the *myTokenTypes* variable to an array containing the local names of the configured token providers:

    ```
    myTokenTypes = AdminTask.listSTSConfiguredTokenTypes()
    print myTokenTypes
    ```
  - Display the non-custom properties for the default token provider.

    The following command returns a java.util.Properties instance that contains the values for each non-custom property for the default token provider stored in the *myDefaultTokenType* variable.

    ```
    AdminTask.querySTSTokenTypeConfigurationDefaultProperties(myDefaultTokenType)
    ```

    To use this command to query a specific token provider, use the following Jython syntax:

    ```
    AdminTask.querySTSTokenTypeConfigurationDefaultProperties('"Security Context Token"')
    ```
  - Display a properties object containing all custom properties for a token provider configuration.

    The following command returns a java.util.Properties instance that contains the values for each of the custom properties for the token provider stored in the *myDefaultTokenType* variable.

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties(myDefaultTokenType)
```

To use this command to query a specific token provider, use the following Jython syntax:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('"Security Context Token"')
```

- Use the following command examples to query endpoint target configurations and security constraints for endpoint targets.
  - Display each uniform resource identifier (URI) for each assigned endpoint.

    The following command sets the *allMyURIs* variable to an array containing the URIs for each assigned endpoint:

    ```
    allMyURIs = AdminTask.listSTSAssignedEndpoints()
    print allMyURIs
    ```

  - Display the token provider that is assigned to a specific endpoint URI.

    The following command sets the *myTokenType* variable to the name of the token provider that is assigned to the `http://myserver.mysom.com:9080/Example` endpoint URI:

    ```
    myTokenType = AdminTask.querySTSEndpointTokenType('http://myserver.mysom.com:9080/Example')
    print myTokenType
    ```

What to do next

Use the wsadmin tool to manage and edit token provider and endpoint configurations.

Managing existing token providers using wsadmin scripting

You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the trust service. Use this topic to modify token provider configuration data, and to add custom properties.

Before you begin

You must have an existing token provider configured in the trust service.

About this task

Use the commands in the STSManagement group of the AdminTask object to modify existing configuration data.

Use the updateSTSTokenTypeConfiguration command to update existing properties for a specific token provider configuration. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.

This topic includes examples for modifying existing non-custom configuration data.

Procedure

1. Determine the token provider configuration to edit.

   Enter the following command to view the list of names of the configured token providers:

```
AdminTask.listSTSConfiguredTokenTypes()
```

2. Review the current configuration data for the token provider configuration to edit.

   Enter the following command to view a Properties object containing all non-custom configuration data for the Security Context Token token provider:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('Security Context Token')
```

3. Update the token provider configuration with new configuration data.

   Determine which parameters to update in your configuration, using the following table as a reference:

Table 117. Command parameters. Run the updateSTSTokenTypeConfiguration command with parameters to update the token provider configuration.

Parameter	Data type
LocalName Specifies the unique token provider name as the target object of the command.	String, required
-HandlerFactory Specifies the configuration class name, including package information.	String, required
-URI Specifies the unique token type schema URI.	String, required
-lifetimeMinutes Specifies the amount of time, in minutes, that the token is valid.	Integer, optional Default: 120 (minutes) Minimum: 10 (minutes)
-renewalWindowMinutes Specifies the amount of time after the token expires during which the token can be renewed.	Integer, optional Default: 120 (minutes) Minimum: 10 (minutes)
-postdatable Set to `true` to specify that tokens of the token provider are valid at a later time. Tokens can be created with or without a future start time.	Boolean, optional Default: false
-distributedCache (deprecated) Set to `true` to enable distributed cache. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.	Boolean, optional Default: false
-renewableAfterExpiration Set to `true` to specify that tokens of the token provider are renewable after expiration.	Boolean, optional Default: false
-tokenCacheFactory (deprecated) Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter.	String, optional Default: `com.ibm.ws.wssecurity.platform.websphere.trust.server.sts.ext.cache.STSTokenCacheFactoryImpl`

Use the **updateSTSTokenTypeConfiguration** command to update the configuration data for the `Security Context Token` token provider. The following example changes the time that the token is valid from 60 minutes to 100 minutes, disables token renewal after expiration, and enables distributed caching:

```
AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-lifetimeMinutes 100
-renewableAfterExpiration false -distributedCache true]')
```

The command returns a message indicating the success or failure of the operation.

4. Save your configuration changes.

   Use the following command to save your changes:

```
AdminConfig.save()
```

5. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the application server:

```
AdminTask.refreshSTS()
```

Adding and removing token provider custom properties using wsadmin scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to administer the trust service. Use this topic to set internal system configuration properties for your token provider configuration by adding or removing custom properties.

Before you begin

You must have an existing token provider configured for the trust service.

About this task

Use custom properties to set internal system configuration properties and specify these properties using the customProperties parameter. Custom properties are arbitrary name and value pairs of data, where the name can be a property key or a class implementation, and where the value might be a string or Boolean value. Use this topic and the commands in the STSManagement group for the AdminTask object to add or remove custom properties from your configuration with the Jython scripting language.

Procedure

- Add new custom properties to a specific token provider configuration.

  Use the **updateSTSTokenTypeConfiguration** command to add or update custom properties to your token provider configuration. Do not use the **updateSTSTokenTypeConfiguration** command to remove custom properties. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.

  1. Launch a scripting command. To learn more, see the starting the wsadmin scripting client information.
  2. Determine the token provider configuration to edit.

     Enter the following command to view a list of the names for each configured token provider:

```
AdminTask.listSTSConfiguredTokenTypes()
```

  3. Review the configured custom properties for the token provider of interest.

     Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
```

  4. Add custom properties to the token provider configuration.

     Use the **updateSTSTokenTypeConfiguration** command to add the configuration data for the `Security Context Token` token provider. Use the following example to add the `com.ibm.ws.security.webChallengeIfCustomSubjectNotFound` custom property with a value of `false` and the `com.ibm.ws.security.defaultLoginConfig` custom property with a value of `system.DEFAULT` to the configuration:

```
AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-customProperties
[[com.ibm.ws.security.webChallengeIfCustomSubjectNotFound false]
[com.ibm.ws.security.defaultLoginConfig system.DEFAULT]] ]')
```

     The command returns a message indicating the success or failure of the operation.

  5. Save your configuration changes.

     Use the following command to save your changes:

```
AdminConfig.save()
```

  6. Reload the modified configuration changes.

     Use the following command to force the trust service to reload your modified configuration without restarting the application server.

     ```
     AdminTask.refreshSTS()
     ```

- Edit custom properties for a specific token provider configuration.

  1. View configured custom properties for the token provider of interest.

     Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
```

  2. Modify the configuration data for the token provider of interest.

Use the **updateSTSTokenTypeConfiguration** command to modify the existing configuration data for the *Security Context Token* token provider. This example specifies that the *Security Context Token* token provider configuration includes the `com.ibm.ws.security.webChallengeIfCustomSubjectNotFound` custom property with a value of `false` and the `com.ibm.ws.security.defaultLoginConfig` custom property with a value of `system.DEFAULT`. Use the following command to change the value of the `com.ibm.ws.security.defaultLoginConfig` custom property from `system.DEFAULT` to `system.CUSTOM`, and does not change any other configured custom properties:

```
AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-customProperties
[[com.ibm.ws.security.defaultLoginConfig system.CUSTOM]]]')
```

The command returns a message indicating the success or failure of the operation.

3. Save your configuration changes.

   Use the following command to save your changes:

```
AdminConfig.save()
```

4. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the application server:

```
AdminTask.refreshSTS()
```

- Remove custom properties from token provider configurations.
  1. View configured custom properties for the token provider of interest.

     Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
```

  2. Delete the custom property from the token provider configuration.

     Use the **deleteSTSTokenTypeConfigurationCustomProperties** command to delete custom properties from your configuration. Specify the names of the custom properties to remove using the propertyNames parameter. If the specified name does not exist in the configuration, no configuration changes are made. The following command removes the `com.ibm.ws.security.webChallengeIfCustomSubjectNotFound` and `com.ibm.ws.security.defaultLoginConfig` custom properties from the *Security Context Token* token provider configuration:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('Security Context Token',
'[-propertyNames com.ibm.ws.security.webChallengeIfCustomSubjectNotFound
com.ibm.ws.security.defaultLoginConfig]')
```

     The command returns a message indicating the success or failure of the operation.

  3. Save your configuration changes.

     Use the following command to save your changes:

```
AdminConfig.save()
```

  4. Reload the modified configuration changes.

     Use the following command to force the trust service to reload your modified configuration without restarting the service:

```
AdminTask.refreshSTS()
```

Associating token providers with endpoint services (targets) using wsadmin scripting

You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the association of endpoints and tokens. Use this topic to query, assign, and unassign the association of a token provider with an endpoint Uniform Resource Identifier (URI).

Before you begin

Before you can assign and manage endpoint configurations, at least one token provider configuration and a web service must exist.

About this task

Use the STSManagement group of commands to specify a custom service endpoint Uniform Resource Identifier (URI) and to assign and unassign the association of trust service token providers with endpoint configurations. Complete the steps in this topic to query the trust service for the existing endpoint configuration, associate the default token with an endpoint, and unassociate a token from an endpoint. You can perform these steps in any order.

Procedure

- Associate a token with a specific endpoint.
  1. View a list of all endpoint URIs that are currently associated with a token provider.

     Before invoking changes on your endpoint configurations, use the following listSTSAssignedEndpoints command to examine your current settings:

     ```
     AdminTask.listSTSAssignedEndpoints()
     ```

     If the endpoint of interest is currently associated with a token, do not use the assignSTSEndpointTokenType command. To update the token that is associated with the endpoint, use the updateSTSEndpointTokenType command in the next step.
  2. Associate a token with an endpoint.

     Use the assignSTSEndpointTokenType command to specify the token to issue for access to a specific endpoint. You do not need to specify the name of the token provider to assign if the token provider is set as the default configuration. For example, the following command assigns the `Security Context Token` default token to the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

     ```
     AdminTask.assignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog')
     ```

     If `Security Context Token` is not the default token provider, use the following command:

     ```
     AdminTask.assignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog',
     '-LocalName Security Context Token')
     ```

     The command returns a message indicating the success of the operation.
  3. Save your configuration changes.

     Use the following command to save your changes:

     ```
     AdminConfig.save()
     ```
  4. Reload the modified configuration changes.

     Use the following command to force the trust service to reload your modified configuration without restarting the application server:

     ```
     AdminTask.refreshSTS()
     ```
- Disassociate a token from an endpoint.
  1. Examine the current endpoint configuration.

     Use the listSTSAssignedEndpoints to view a list of each endpoint URI with assigned token providers, as the following example describes:

     ```
     AdminTask.listSTSAssignedEndpoints()
     ```

     The following sample output is displayed:

     ```
     'http://www.mycompany.com:8080/Ecommerce/Catalog'
     ```
  2. Choose the endpoint to edit.

Use the querySTSEndpointTokenType to return the token provider associated with the endpoint of interest. Enter the following command to view the token provider associated with the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

```
AdminTask.querySTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog')
```

The following sample output is displayed:

```
'Security Context Token'
```

3. Disassociate the token type from the endpoint.

   Use the **unassignSTSEndpointTokenType** command to disassociate the token provider and endpoint configuration. The following command removes the `Security Context Token` token provider that is associated with the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

   ```
   AdminTask.unassignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog',
   '-LocalName Security Context Token')
   ```

   The command returns a message indicating the success of the operation.

4. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

5. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the service:

   ```
   AdminTask.refreshSTS()
   ```

STSManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the STSManagement group can be used to manage and query trust service token provider configurations and endpoint configurations.

The STSManagement command group contains commands that allow you to configure existing token providers, assign token providers to endpoints, and modify general trust service configuration data. The commands in this group that perform configuration changes require that you execute the save command to commit the changes. No configuration changes are made if an exception is created when executing a command.

Use the following commands to modify and query token provider configurations:

Use the following commands to assign, unassign, and query endpoint configurations:

- “updateSTSEndpointTokenType ” on page 700

Use the following commands to add, edit, delete, and list properties of the trust service:
- “addSTSProperty” on page 701
- “deleteSTSProperty” on page 701
- “editSTSProperty” on page 702
- “listSTSProperties” on page 702

Use the following command to force the trust service to reload your modified configuration without restarting the application server:
- “refreshSTS ” on page 703

createSTSTokenTypeConfiguration

The createSTSTokenTypeConfiguration command is used to create a token provider configuration.

Target object

Specify the LocalName object, which is used as an identifier for the various configurations. The value for the LocalName object must be unique.

Required parameters

-URI
The URI of the token provider. This value must be unique across all configuration token type URIs. (String, required)

-HandlerFactory
Provide the fully qualified class name of an implementation of the `org.eclipse.higgins.sts.IObjectFactory` interface. (String, required)

Optional parameters

-lifetimeMinutes
Specifies the maximum lifetime to assign to an issued token provider. The default value is 120 minutes. (Integer, optional)

-distributedCache
Specifies whether to enable or disable distributed cache. Specify `true` to enable distributed cache capability. The default value is `false`. If you specify this option, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for this parameter for custom tokens. (Boolean, optional)

-tokenCacheFactory
Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter. (String, optional).

Return value

The command returns a success or failure message.

Batch mode example usage
- Using Jython string:

```
AdminTask.createSTSTokenTypeConfiguration('myTokenType', '[-HandlerFactory
 test.ibm.samples.myTokenType -URI http://ibm.com/tokens/schema/myTokenType]')
```
- Using Jython list:

```
AdminTask.createSTSTokenTypeConfiguration('myTokenType', ['-HandlerFactory',
 'test.ibm.samples.myTokenType', '-URI', 'http://ibm.com/tokens/schema/myTokenType'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createSTSTokenTypeConfiguration('-interactive')
```

deleteSTSTokenTypeConfigurationCustomProperties

The deleteSTSTokenTypeConfigurationCustomProperties command is used to remove custom properties from a token provider configuration.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

-propertyNames
Specify the names of the custom properties to delete from the configuration. If any of the specified properties do not exist in your configuration, you will receive an error message. (String[], optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('myTokenType', '[-propertyNames
 com.ibm.ws.security.webChallengeIfCustomSubjectNotFound com.ibm.ws.security.defaultLoginConfig]')
```

- Using Jython list:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('myTokenType', ['-propertyNames',
 'com.ibm.ws.security.webChallengeIfCustomSubjectNotFound com.ibm.ws.security.defaultLoginConfig'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('-interactive')
```

listSTSConfiguredTokenTypes

The listSTSConfiguredTokenTypes command is used to list the local names of all configured token providers.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the local names of all configured token providers.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSConfiguredTokenTypes()
```

Interactive mode example usage

- Using Jython:

```
AdminTasklistSTSConfiguredTokenTypes('-interactive')
```

querySTSDefaultTokenType

The querySTSDefaultTokenType command is used to determine the local name of the default token provider.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the local name of the default token provider.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSDefaultTokenType()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSDefaultTokenType('-interactive')
```

querySTSTokenTypeConfigurationDefaultProperties

The querySTSTokenTypeConfigurationDefaultProperties command is used to query the trust service for the non-custom properties of a token provider.

Target object

Specify the LocalName object of the token provider to query.

Required parameters

None

Optional parameters

None

Return value

The command returns a `java.util.Properties` instance which contains the values of the non-custom properties. Non-custom properties include URI, HandlerFactory, lifetimeMinutes, distributedCache, postdatable, renewableAfterExpiration, and renewalWindowMinutes.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('-interactive')
```

querySTSTokenTypeConfigurationCustomProperties

The querySTSTokenTypeConfigurationCustomProperties command is used to query the trust service.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

None

Return value

The command returns a `java.util.Properties` instance containing the values of the custom properties.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('-interactive')
```

setSTSDefaultTokenType

The setSTSDefaultTokenType command is used to set the default token provider for the trust service.

Target object

Specify the LocalName object of the token provider as default.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.setSTSDefaultTokenType('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setSTSDefaultTokenType('-interactive')
```

updateSTSTokenTypeConfiguration

The updateSTSTokenTypeConfiguration command is used to update configuration data for a token provider. All parameters are optional. The parameters that are specified are updated in the configuration if the property already exists. If the property does not exist, it is added to the configuration. To remove custom properties, use the **deleteSTSTokenTypeConfigurationCustomProperties** command.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

-URI
The URI of the token provider. This value must be unique across all configuration token type URIs. (String, optional)

-HandlerFactory
Provide the fully qualified class name of an implementation of the `org.eclipse.higgins.sts.utilities.IObjectFactory` interface. (String, optional)

-lifetimeMinutes
The maximum lifetime to assign to an issued token provider. The default value is 120 minutes. (Integer, optional)

-distributedCache
Specifies whether to enable or disable distributed cache. Specify `true` to enable distributed cache capability. The default value is `false`. If you specify this option, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for this parameter for custom tokens. (Boolean, optional)

-postdatable
Set the value of this parameter to `true` to allow tokens of this token provider to be valid starting at a future time. The default value is false. (Boolean, optional)

-renewableAfterExpiration
Set the value of this parameter to `true` to allow tokens of this token provider to be renewable after expiration. The default value is false. (Boolean, optional)

-renewableWindowMinutes
Provide the number of minutes after a token has expired that a token of this token provider can be renewed. If this specified time has elapsed after expiration, then the token will no longer be available for renewal. The default value is 120 minutes. (Integer, optional)

-tokenCacheFactory
Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter. (String, optional).

-customProperties
Provide any additional custom properties. (java.util.Properties, optional).

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSTSTokenTypeConfiguration('myTokenType', '[-lifetimeMinutes 100
 -renewableAfterExpiration false -distributedCache true]')
```

- Using Jython list:

```
AdminTask.updateSTSTokenTypeConfiguration('myTokenType', ['-lifetimeMinutes', '100', '
 -renewableAfterExpiration', 'false', '-distributedCache', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSTSTokenTypeConfiguration('-interactive')
```

removeSTSTokenTypeConfiguration

The removeSTSTokenTypeConfiguration command removes a token provider configuration.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.removeSTSTokenTypeConfiguration('myTokenType')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeSTSTokenTypeConfiguration ('-interactive')
```

assignSTSEndpointTokenType

The assignSTSEndpointTokenType command is used to give a token provider when a specific endpoint is accessed.

Target object

Specify the endpointURI object of the endpoint to assign a given token provider. If the specified endpoint has already been assigned a token provider, you will receive an error message.

Required parameters

None

Optional parameters

-LocalName
Specify the local name of the token provider to assign to the specified endpoint. If the token provider configuration does not exist, you will receive an error message. If this parameter is not specified, the default token provider is used. (String, optional)

-issuer
Specify the URI of the issuer that specifies the token provider to issue. This value can be null. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.assignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', '[-LocalName
 tokenType1]')
```

- Using Jython list:

```
AdminTask.assignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.assignSTSEndpointTokenType ('-interactive')
```

listSTSAssignedEndpoints

The listSTSAssignedEndpoints command is used to list the URIs of assigned endpoints.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the URIs of all assigned endpoints.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSAssignedEndpoints()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSAssignedEndpoints ('-interactive')
```

listSTSEndpointTokenTypes

The listSTSEndpointTokenTypes command is used to query the Trust Service for the token provider assigned to a specific endpoint.

Target object

Specify the endpointURI object of the endpoint to query. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

None

Optional parameters

None

Return value

The command returns the local name of the token provider assigned to the specified endpoint.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSEndpointTokenTypes()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSEndpointTokenTypes ('-interactive')
```

unassignSTSEndpointTokenType

The unassignSTSEndpointTokenType command is used to unassign an endpoint from its token provider.

Target object

Specify the endpointURI object of the endpoint to unassign from a given token provider. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

-LocalName
Specify the local name of the token provider configuration to unassign from the specified endpoint. (String, required)

Optional parameters

-issuer
Specify the URI of the issuer in the token provider assignment to remove. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.unassignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/','[-LocalName
 tokenType2]')
```

- Using Jython list:

```
AdminTask.unassignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType2'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unassignSTSEndpointTokenType ('-interactive')
```

updateSTSEndpointTokenType

The updateSTSEndpointTokenType command is used to assign a different token provider to a specified endpoint.

Target object

Specify the endpointURI object of the endpoint to update. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

-LocalName
Specify the local name of the token provider to assign to the specified endpoint. If the token provider configuration does not exist, you will receive an error message. If this parameter is not specified, the default token provider is used. (String, optional)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', '[-LocalName
 tokenType2]')
```

- Using Jython list:

```
AdminTask.updateSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType2'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSTSEndpointTokenType('-interactive')
```

addSTSProperty

The addSTSProperty command adds a new property for the trust service.

Target object

Specify a unique name for the new property (string, required).

Required parameters

-propertyValue
Specifies the value of the property to add. (String, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addSTSProperty('pluginSCTVersion', '[-propertyValue 2.0]')
```

- Using Jython list:

```
AdminTask.addSTSProperty('pluginSCTVersion', ['-propertyValue', '2.0'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addSTSProperty('-interactive')
```

deleteSTSProperty

The deleteSTSProperty command deletes an existing property from the trust service.

Target object

Specify the name of the property to delete.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.deleteSTSProperty('pluginSCTVersion')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSTSProperty('-interactive')
```

editSTSProperty

The editSTSProperty command modifies an existing property for the trust service.

Target object

Specify the name of the property to edit. (String, required)

Required parameters

-propertyValue
Specifies the new value for the property of interest. (String, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.editSTSProperty('pluginSCTVersion', '[-propertyValue 2.1]')
```

- Using Jython list:

```
AdminTask.editSTSProperty('pluginSCTVersion', ['-propertyValue', '2.1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.editSTSProperty('-interactive')
```

listSTSProperties

The listSTSProperties command lists all existing properties and their corresponding values for the trust service.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a java.util.Properties instance that contains the names and values of the properties.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSProperties()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSProperties('-interactive')
```

refreshSTS

The refreshSTS command refreshes your trust service configuration changes without restarting the application server.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.refreshSTS()
```

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - IBM i

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations.

app_client_root
The default installation root directory for the Application Client for IBM WebSphere Application Server is the `/QIBM/ProdData/WebSphere/AppClient/V8/client` directory.

app_client_user_data_root
The default Application Client for IBM WebSphere Application Server user data root is the `/QIBM/UserData/WebSphere/AppClient/V8/client` directory.

app_client_profile_root
The default Application Client for IBM WebSphere Application Server profile root is the `/QIBM/UserData/WebSphere/AppClient/V8/client/profiles/`*`profile_name`* directory.

app_server_root
The default installation root directory for WebSphere Application Server Network Deployment is the `/QIBM/ProdData/WebSphere/AppServer/V8/ND` directory.

java_home

Table 118. Root directories for supported Java Virtual Machines.

This table shows the root directories for all supported Java Virtual Machines (JVMs).

JVM	Directory
32–bit IBM Technology for Java	`/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit`
64–bit IBM Technology for Java	`/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit`

plugins_profile_root
The default Web Server Plug-ins profile root is the `/QIBM/UserData/WebSphere/Plugins/V8/webserver/profiles/`*`profile_name`* directory.

plugins_root
The default installation root directory for Web Server Plug-ins is the `/QIBM/ProdData/WebSphere/Plugins/V8/webserver` directory.

plugins_user_data_root
The default Web Server Plug-ins user data root is the `/QIBM/UserData/WebSphere/Plugins/V8/webserver` directory.

product_library
product_lib
This is the product library for the installed product. The product library for each Version 8.0 installation on the system contains the program and service program objects (similar to `.exe`, `.dll`, `.so` objects) for the installed product. The product library name is `QWAS8`*`x`* (where *x* is A, B, C, and so on). The product library for the first WebSphere Application Server Version 8.0 product installed on the system is `QWAS8A`. The *`app_server_root`*`/properties/product.properties` file contains the value for the product library of the installation, `was.install.library`, and is located under the *`app_server_root`* directory.

profile_root
The default directory for a profile named *profile_name* for WebSphere Application Server Network Deployment is the `/QIBM/UserData/WebSphere/AppServer/V8/ND/profiles/`*`profile_name`* directory.

shared_product_library
The shared product library, which contains all of the objects shared by all installations on the system, is `QWAS8`. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

user_data_root
The default user data directory for WebSphere Application Server Network Deployment is the `/QIBM/UserData/WebSphere/AppServer/V8/ND` directory.

The `profiles` and `profileRegistry` subdirectories are created under this directory when you install the product.

web_server_root
The default web server path is /www/*web_server_name*.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index