Using the Configuration Web Services

This chapter describes the environment that you must set up to use the Configuration web service and explains how to use it.

The Configuration web services are implemented as REST interfaces over HTTPS. There is no HTTP support.

Configuring REST web services are available on all ACS servers in the deployment, but only the ACS primary instance provides the full service that supports read and write operations. Secondary ACS instances provide read only access to the configuration data.

The Monitoring and Report Viewer displays the messages and audit logs for all REST activities.

**Enabling the REST Web Interface on ACS CLI**

You must enable the web interface on ACS before you can use the REST web service. To enable the web interface on ACS, from the ACS CLI, enter:

```
acs config-web-interface rest enable
```


**Viewing the Status of the REST Web Interface from ACS CLI**

To view the status of the web interface, from the ACS CLI, enter:

```
show acs-config-web-interface
```


Application that interacts with ACS configuration REST service may use any administrator account to authenticate to the REST service. Authorization for the used account should be set to allow all activities done by the REST client.

**Supported Configuration Objects**

The Rest PI in ACS provides services for configuring ACS and it is organized for each configuration feature. In ACS 5.3, the following two subsets of the ACS configuration are supported.

- Common configuration objects
- Identity configuration objects
Table 4-1 lists the supported configuration objects.

### Table 4-1 Supported Configuration Objects

<table>
<thead>
<tr>
<th>Feature</th>
<th>Main Supported Classes</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>Attribute Info</td>
<td>Also known as dynamic attributes or AV pair. Attribute Info is composed within Protocol User.</td>
</tr>
<tr>
<td>ACS Version</td>
<td></td>
<td>Supports Get method only.</td>
</tr>
<tr>
<td>Service Location</td>
<td></td>
<td>Supports getall method only. It allows to find the ACS instance that serve as primary and the ACS instance that provide Monitoring and Troubleshooting Viewer.</td>
</tr>
<tr>
<td>Error Message</td>
<td></td>
<td>Supports getall method only. It allows to retrieve all ACS message codes and message texts that are used on the REST Interface.</td>
</tr>
<tr>
<td>Identity</td>
<td>Protocol User</td>
<td>Full CRUD (Create, Read, Update, and Delete) and query support.</td>
</tr>
<tr>
<td>Identity Group</td>
<td></td>
<td>Full CRUD and query support. Query is used to retrieve subgroups of a specific node. The list of users for each group is fetched by querying on the users.</td>
</tr>
</tbody>
</table>

This section contains:

- Identity Groups, page 4-2
- Attribute Info, page 4-3
- Group Associations, page 4-3

### Identity Groups

Identity Group object is used to manipulate nodes on the Identity Group hierarchy. The group name defines the full path of the node within the hierarchy. When you add a new node, you should be aware that the name of the node (which includes the full path) specifies where in the hierarchy the node should be attached. For example:

- All Groups:CDO:PMBU
- All Groups:CDO
- All Groups:CDO:PMBU:ACS-Dev

**Note**

You must create the upper level hierarchy (parent node) and then create the leaf node. For example: To create the hierarchy, All Groups:US:WDC; we must create All Groups:US and then go ahead creating the next level in hierarchy.
In order to retrieve child of certain group you can set a filter as “start with All groups:CDO”.

**Attribute Info**

The AttributeInfo structure is an array of pairs of attribute names and attribute values. The attribute name refers to the user dictionary, where the definition of the attribute, such as value type, can be found. The value of the attribute must conform with the dictionary definition.

The following is an example of JAVA representation for a user that has two attributes:

```java
User user = new User();
    user.setDescription(description);
    user.setPassword(password);
    user.setName(userName);
    user.setAttributeInfo(new AttributeInfo[]{
        new AttributeInfo("Department","Dev"),
        new AttributeInfo("Clock","10 Nov 2008 12:12:34")
    });
```

**Group Associations**

The REST Interface schema shows the association of the user to the Identity group, as a group name property on the user object.

Here is an example of associating user to an identity group:

```java
User user = new User();
    user.setIdenityGroupName("IdentityGroup:All Groups:Foo");
    user.setDescription(description);
    user.setPassword(password);
    user.setName(userName);
```

**Query Object**

The REST Interface schema exposes a query object to define criteria and other query parameters. The query object is used for users and identity groups.

The query object includes parameters that apply to:

- Filtering, page 4-3
- Sorting, page 4-4
- Paging, page 4-5

**Filtering**

You can use the query object to retrieve a filtered result set. You can filter users or identity groups, based on the following criteria:

- Simple condition—Includes property name, operation, and value. For example, name STARTS_WITH "A".

The following operations are supported for filtering:

- CONTAINS
• DOES_NOT_CONTAIN
• ENDS_WITH
• EQUALS
• NOT_EMPTY
• NOT_EQUALS
• STARTS_WITH

• And condition—Includes set of simple conditions. All simple condition must be evaluated to be True in order for the and condition to be matched.

Here is the XML based example for the “And” filter.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"/>
<ns2:query xmlns:ns2="query.rest.mgmt.acs.nm.cisco.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:type="ns2:AndFilter">
<criteria>
  <simpleFilters>
    <propertyName>name</propertyName>
    <operation>STARTS_WITH</operation>
    <value>user</value>
  </simpleFilters>
  <simpleFilters>
    <propertyName>name</propertyName>
    <operation>ENDS_WITH</operation>
    <value>1</value>
  </simpleFilters>
</criteria>
<numberOfItemsInPage>100</numberOfItemsInPage>
<startPageNumber>1</startPageNumber>
</ns2:query>
```

Here is a Java based example for the ‘And’ filter:

```java
Query query = new Query();
query.setStartPageNumber(1);
query.setNumberOfItemsInPage(100);

SimpleFilter simpleFilter = new SimpleFilter();
simpleFilter.setOperation(FilterOperation.STARTS_WITH);
simpleFilter.setPropertyName("name");
simpleFilter.setValue("user");

SimpleFilter simpleFilter1 = new SimpleFilter();
simpleFilter1.setOperation(FilterOperation.ENDS_WITH);
simpleFilter1.setPropertyName("name");
simpleFilter1.setValue("1");

AndFilter andFilter = new AndFilter();
andFilter.setSimpleFilters(new SimpleFilter[]{ simpleFilter, simpleFilter1 });

query.setCriteria(andFilter);
```

### Sorting

You can use the query object to sort the results. You can sort based on the following criteria:

• One property to sort by
Paging

You can set the query object with the following paging parameters:
- Page number, which is the requested page
- Number of objects in a page

Paging is stateless. That is, the required page is calculated from scratch for every request. This means that paging could skip objects or return them twice, in case objects were added or deleted concurrently.

Request Structure

ACS REST request is composed of:
- URL
- HTTP method
- Content—Includes ACS objects if applicable to the requested method. The ACS objects are represented in XML.

URL Path

URL includes:
- Service name: Rest
- Package name: Identity or Common
- Object Type: User, Identity Group, and so on
- Object Identifier are valid with GET and DELETE methods
- Operation name is required for operations other than CRUD such as query.

Table 4-2 lists the URLs for each object.

Object Identifiers

Objects are identified by name or by object ID. Basic object key is the object name. You can also use Object ID for GET and Delete method. For POST and PUT, the method gets the object itself that includes the identifiers.

You can specify identifier on the URL in the following ways:
- Name as the key — Rest/{package}/{Object Type}/name/{name}
- Object ID as the key — Rest/{package}/{Object Type}/id/{id}
- For single instance per object type, no key is required — For example: REST/Common/AcsVersion
**HTTP Methods**

HTTP methods are mapped to configuration operations (CRUD - Create, Read, Update, and Delete).

The common intrinsic methods are not specified within the URL, and are determined by the HTTP request method. In other cases, you need to add the configuration operation into the URL. HTTP methods are mapped to ACS operations:

- **HTTP GET**—View an object or multiple objects
- **HTTP POST**—Create a new object
- **HTTP DELETE**—Delete a object
- **HTTP PUT**—Update an existing object. PUT is also used to invoke extrinsic methods (other than CRUD).

When HTTP PUT method is used for operations other than CRUD, the URL specifies the required operation. This is also used to distinguish the message from PUT method for update. The keyword “op” is included in the URL as follows:

```
Rest/{package}/{ObjectType}/op/{operation}
```

For example, /Rest/Identity/IdentityGroup/op/query

Table 4-3 describes the primary ACS REST methods and their mapping to HTTP messages.

**Table 4-3**  **HTTP Method Summary**

<table>
<thead>
<tr>
<th>Function</th>
<th>HTTP Method</th>
<th>URL</th>
<th>Request content</th>
<th>Response on Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAll</td>
<td>GET</td>
<td>/{ObjectType}</td>
<td>None</td>
<td>Collection of Objects</td>
</tr>
<tr>
<td>getName</td>
<td>GET</td>
<td>/{ObjectType}/name</td>
<td>None</td>
<td>An Object</td>
</tr>
<tr>
<td>getId</td>
<td>GET</td>
<td>/{ObjectType}/id/{id}</td>
<td>None</td>
<td>An Object</td>
</tr>
</tbody>
</table>
Chapter 4 Using the Configuration Web Services

Table 4-3 HTTP Method Summary (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>HTTP Method</th>
<th>URL</th>
<th>Request content</th>
<th>Response on Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>POST</td>
<td>/{ObjectType}</td>
<td>Object</td>
<td>Rest Response Result, which includes Object ID.</td>
</tr>
<tr>
<td>delete</td>
<td>DELETE</td>
<td>/{ObjectType}/name/{name} (^1)</td>
<td>None</td>
<td>Rest Result</td>
</tr>
<tr>
<td>delete</td>
<td>DELETE</td>
<td>/{ObjectType}/id/{id}</td>
<td>None</td>
<td>Rest Result</td>
</tr>
<tr>
<td>update(^2)</td>
<td>PUT</td>
<td>/{ObjectType}</td>
<td>Object</td>
<td>Rest Result</td>
</tr>
<tr>
<td>Query</td>
<td>PUT</td>
<td>/{ObjectType}/op/query</td>
<td>QueryObject</td>
<td>List of Objects</td>
</tr>
</tbody>
</table>

1. Names in the URL are full names. ACS REST services does not support wildcards or regular expressions.
2. Update method replaces the entire object with the object provided in the request body, with the exception of sensitive properties.

Note For the responses on failure, see ACS REST Result.

Response Structure

The response to Rest request is a standard HTTP response that includes HTTP status code and other data returned by web servers. In addition, the response can include the ACS Rest Result object or ACS configuration objects according to the type of request.

You should check the HTTP status code to know the type of objects expected in the response body.

- For 4xx HTTPS status codes except for 401 and 404, REST result Object is returned.
- For 5xx status codes other than 500, the message content includes a text that describe the server error.
- For 500 HTTP status code, REST result is returned.
- For 200 and 201 HTTP status code, objects per the specific method or object type is returned.
- For 204 HTTP status code, no object is returned.

HTTP Status Codes

ACS returns the following types of status codes:

- 2xx for success
- 4xx for client errors
- 5xx for server errors
ACS does not return the following types of status codes:

- 1xx
- 3xx

The HTTP status code is returned within the HTTP response headers as well as within the REST result object.

Table 4-4 lists the HTTP status codes that are returned by ACS.

Table 4-4  Usage of HTTP Status Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Message</th>
<th>Usage in ACS</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Ok</td>
<td>Successful Get, create and query</td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>OK with no content</td>
<td>Successful delete and update</td>
<td>No data is returned in the response body.</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>Request errors: Object validation failure, XML syntax error, and other error in request message</td>
<td>The request contains bad syntax or cannot be executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For example, if you try to create an object with a name that already exists, the object validation fails. Detailed reasons can be found in the REST result object.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>Authentication Failure/ Time outs</td>
<td>Similar to 403 error, but specifically for use when authentication failed or credentials are not available.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>ACS is a secondary and cannot fulfill the request or operation is not allowed per administrator authorizations.</td>
<td>The request was valid, but the server refuses to respond to it. Unlike a 401 error, authenticating will make no difference. Also, this error is displayed when an non-read request was send to a secondary instance.</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>For cases where the URL is wrong or the REST Service is not enabled.</td>
<td></td>
</tr>
<tr>
<td>410</td>
<td>Gone</td>
<td>A resource is not available anymore</td>
<td>A request was made for an object that does not exist. For example, deleting an object that does not exist.</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
<td>For any Server error that has no specific HTTP Code.</td>
<td></td>
</tr>
</tbody>
</table>
ACS REST Result

The HTTP response for a REST request includes either requested objects or REST result object. See Table 4-3 for details. ACS result includes:

- HTTP status code
- HTTP status text
- ACS message code
- ACS message
- Object ID for successful CREATE method

returned objects

ACS returns objects for GET method and for query operation. The type of returned object is determined by the request URL.

When a GET method returns multiple objects, these are included in the response. If the returned list is too long, you should use filtering or paging options.

WADL File

The WADL files contain the object structure (schema) and the methods for every object.

The WADL files are mainly documentation aids. You cannot generate client applications using WADL files.

The WADL file structure is according to W3C specification. For more information, see http://www.w3.org/Submission/wadl/

To download the WADL files:

Step 1 From the ACS user interface, go to System Administration > Downloads > Rest Service

Step 2 Under ACS Rest Service WADL files, click Common or Identity and save the files to your local drive.

Schema File

ACS is shipped with three XSD files that describe the structure of the objects supported on ACS 5.3 REST interfaces.

The three XSD files are:

- Common.xsd, that describes the following objects:
  - Version
  -AttributeInfo
  - Error Message
  - ResultResult, RestCreateResult
Sample Code

ACS provides sample code for client application to help you develop an application that interacts with ACS REST Interface. The sample code can be downloaded in the same way as WADL and schema files. The sample code is based on Apache HTTP Client http://hc.apache.org/httpcomponents-client-ga/index.html and JAVA code generated by JAXB (xjc command) with the help of the XSD files. It includes sample codes for:

- Get ACS Version
- Get all users
- Get All Service Locations
- Get Filtered list of Users
- Get list of Error messages
- Get User by ID and by name
- Create, Delete, Update user
- Create, Delete, and Update identity group
- Get IdentityGroup by name or ID
- Get sub-tree of IdentityGroups
- Get all Users of an Identity Group

Note

It is highly recommended to generate REST client classes from the XSD files than coding XML or creating it manually.