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
```

For more information on the **acs config-web-interface** command, see *CLI Reference Guide for Cisco Secure Access Control System 5.8*.

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
```

For more information on the **acs config-web-interface** command, see *CLI Reference Guide for Cisco Secure Access Control System 5.8*.

Applications that interact with the 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 that are 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 5.8, the following five subsets of the ACS configuration are supported:

- Common configuration objects
- Users configuration objects, Hosts configuration objects, and Identity group configuration objects.
- Network Device configuration objects
- Device Group configuration objects
- Device Group Type configuration objects
- Maximum User Sessions

Table 1 on page 2 lists the supported configuration objects.

Table 1 Supported Configuration Objects

Feature	Main Supported Classes	Comments
Common	Attribute Info	Also known as dynamic attributes or AV pair. Attribute Info is composed within Protocol User.
	ACS Version	Supports Get method only.
	Service Location	Supports getall method only. It allows you to find the ACS instance that serve as primary and the ACS instance that provide Monitoring and Troubleshooting Viewer.
	Error Message	Supports getall method only. It allows you to retrieve all ACS message codes and message texts that are used on the REST Interface.
Identity	Protocol User	Full CRUD (Create, Read, Update, and Delete) and query support.
	Identity Group	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.
	Internal Host	Full CRUD and query support.
Network Device	Network Device	Full CRUD and query support.
Device Group	Network Device Group	Full CRUD and query support.
Device Group Type	Network Device Group Type	Full CRUD and query support.

This section contains:

Identity Groups

The 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 the child of a 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:

```
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 a user to an identity group:

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

Device Info

The following is an example of a Java representation for the configuration of a network device group and information that is related to creating a network device. It is also a good example of using a GroupInfo object to define device groups.

```
IPSubnetType ip1 = new IPSubnetType();
                ip1.setIpAddress("1.1.1.1-5");
                ip1.setNetMask(32);
                ip1.setIpSubnetExclude(new IPSubnetExcludeType().setIpAddress("1.1.1.3");
IPSubnetType ip2 = new IPSubnetType();
                ip1.setIpAddress("3.3.3.1-5");
                ip1.setNetMask(32);
                ip1.setIpSubnetExclude(new IPSubnetExcludeType().setIpAddress("3.3.3.3");

TACACSConnection tacacsCon = new TACACSConnection();
                tacacsCon.setSharedSecret("secret");
                tacacsCon.setSingleConnect("true");
                tacacsCon.setLegacyTACACS("true");

RADIUSConnection radiusCon = new RADIUSConnection();
                radiusCon.setSharedSecret("secret");
                radiusCon.setPortCoA(1700);
                radiusCon.setKeyWrap(true);
                radiusCon.setKeyEncryption("Key");
                radiusCon.setMessageAuthenticationCodeKey("AuthKey");
                radiusCon.setDisplayedInHex(false);

Device device = new Device();
```

```
                device.setName(deviceName);
                device.setdescription(deviceDescription);
                device.setGroupInfo(new GroupInfo[]{new GroupInfo("All
Locations:Chennai","Location"),new GroupInfo("All Device Types:Router","Device Type"});
                device.setSubnets(new IPSubnetType[]{ip1,ip2});
                device.setTacacsConnection();
                device.setRadiusConnection(radiusCon);
```

To create a single IP without mask or ranges, you need to just create the IPSubnetType and associate the subnet to that object, as shown in the above example.

Note: It is mandatory to provide information about at least one group to which the network device is associated while updating a network device using the REST interface. The network device may be associated with more than one group. The group information that is not provided will automatically be updated with a default value.

Query Object

The REST Interface schema exposes a query object to define criteria and other query parameters. The query object is used for Users, Identity Groups, Network Device, Network Device Groups, Internal Hosts, and Maximum User Sessions.

The query object includes parameters that apply to:

- Filtering, page 4
- Sorting, page 5
- Paging, page 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
  - IS_EMPTY
  - 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 an XML-based example for the “And” filter.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <ns2:query xmlns:ns2="query.rest.mgmt.acs.nm.cisco.com">
            <criteria xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:type="ns2:AndFilter">
                  <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>
```

Note: The XML tag `numberOfItemsInPage` has been changed to `numberofItemsInPage` (from upper case "O" to lower case "o") in patch 3. You need to install patch 3 to get this change reflected in ACS.

Here is a Java-based example for the "And" filter:

```
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
- Direction of sorting (Ascending/Descending)

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, Common, or Network Device
- Object Type: User, Identity Group, and so on
- Object Identifiers are valid with the GET and DELETE methods.
- Operation name is required for operations other than CRUD, such as query.

Table 2 on page 6 lists the URLs for each object.

Object Identifiers

Objects are identified by name or by object ID. The basic object key is the object name. You can also use the Object ID for the GET and DELETE methods. For POST and PUT, the method gets the object itself, which includes the identifiers.

You can specify the identifier on the URL in the following ways:

- Name as the key: Rest/{package}/{ObjectType}/name/{name}
- Object ID as the key: Rest//{package}/{ObjectType}/id/{id}
- For a single instance per object type, no key is required. `For example: REST/Common/AcsVersion`

Table 2 URL Summary Table

Object	URL	Comment
ACS Version	../Rest/Common/AcsVersion	Single object exists
Service Location	../Rest/Common/ServiceLocation	–
Error Message	../Rest/Common/ErrorMessage	–
User	../Rest/Identity/User/.....	For some methods, there is additional data on the URL. See Table 3 on page 7
Identity Group	../Rest/Identity/IdentityGroup/.....	For some methods, there is additional data on the URL. See Table 3 on page 7
Network Device	../Rest/NetworkDevice/Device/.....	For some methods, there is additional data on the URL. See Table 3 on page 7
Device Group	..Rest/NetworkDevice/DeviceGroup/.....	For some methods, there is additional data on the URL. See Table 3 on page 7
Internal Host	../Rest/Identity/Host/.....	For some methods, there is additional data on the URL. See Table 3 on page 7

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 the 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 the 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 3 on page 7 describes the primary ACS REST methods and their mapping to HTTP messages.

Table 3 HTTP Method Summary

Function	HTTP Method	URL	Request content	Response on Success
getAll	GET	/{ObjectType}	None	Collection of Objects
getAllDevices	GET	/{ObjectType}	None	List of all devices with basic information and without Network Device Group information.
getByName	GET	/{ObjectType}/name/{name}[1]	None	An Object
getById	GET	/{ObjectType}/id/{id}	None	An Object
create	POST	/{ObjectType}	Object **Note:** For create, the Object ID property should not be set.	Rest Response Result, which includes Object ID.
delete	DELETE	/{ObjectType}/name/{name}1	None	Rest Result
delete	DELETE	/{ObjectType}/ id/{id}	None	Rest Result
update[2]	PUT	/{ObjectType}	Object	Rest Result
Query	PUT	/{ObjectType}/op/query	QueryObject	List of Objects

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: The GET and DELETE methods using MAC addresses are applicable only for the Host object types. For example, `/Host/macaddress/{macaddress}`. The name attribute in these methods is replaced with MAC addresses. This is because of the Host object, which does not have the name attribute.

Note: For the responses on failure, see ACS REST Result, page 10.

getAllDevices method

ACS 5.8 introduces a light weight REST API method called “getAllDevices” that helps you to get the list of network devices and AAA clients information without the Network Device Groups. This method returns the list of all network devices with basic and minimal information. The basic information includes the name, IP address, description, device id, version, and authentication related details. This new lightweight “getAllDevices” method increases the response time to fetch data from ACS.

When you use the existing “getAll” method to fetch device information from ACS, if you have 10 or 100 devices, it fetches all information including the network device group quickly. But, when you have 10k devices and you try to fetch data using “getAll” method, it takes around 30 minutes to process the request. Therefore, in such cases, you need to use the light weight “getAllDevices” method. The light weight “getAllDevices” method retrieves the limited basic information from ACS quickly. You can use the name or ID of the device and use “getByName” or “getById” methods to get all related information of that device. Now, you can perform the CRUD operations on the objects that are received.

If you are going to use the light weight “getAllDevices” API method, update to the version field in ACS will be done by ACS itself, however, you can add a version field in your custom database and modify it. The version field indicates that the number of times an object is modified. This version field must be increased by one every time in your custom database when you update that object. When you retrieve data using “getAllDevices” method, the version field is also retrieved from ACS. You need to compare the object version in your custom database with the retrieved objects version from ACS. You need to compare the versions in your custom database with the records that are fetched from ACS. If the retrieved version of the object is different from what you have it in your custom database, then you need to update that object using the “getByName” or “getById” API methods.

Multiple Parallel REST requests results in Denial of Service in ACS:

When a primary ACS node receives a REST request, it verifies the administrator credentials and establishes a TCP connection with the REST client and stores the authentication results along with the last login time and replicates the results secondary nodes.

When ACS receives multiple parallel REST requests at the same time, ACS tries to store the authentication results in the database and replicate the results to all secondary nodes. This results in denial of service in ACS.

It is recommended to establish a connection against ACS and use this connection for all the subsequent REST requests to avoid the stability issues.

- If you use Java clients, you can register the client against ACS, establish a TCP connection, and use this connection for all subsequent requests. For more information, see REST client Java class section on the SDK examples of in ACS Web interface (**System Administration > Downloads > REST Service > SDK samples**)
- If you use CURL clients, you must save the cookies from CURL program and use the cookies for all subsequent REST requests.

To save and use CURL cookies, complete the following steps:

1. Execute the following command to save the cookies in cookies.txt file:

```
curl -c cookies.txt https://<ACS_IP_ADRESS>/REST/Common/AcsVersion
```

2. Execute the following command to use the cookies from cookies.txt file

```
curl -b cookies.txt https://<ACS_IP_ADRESS>/REST/Common/AcsVersion
```

Response Structure

The response to REST request is a standard HTTP response that includes the HTTP status code and other data that is 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 find out the type of objects that are 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 text that describe the server error.
- For 500 HTTP status codes, REST result is returned.
- For 200 and 201 HTTP status codes, objects per the specific method or object type are returned.
- For 204 HTTP status codes, 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 on page 9 lists the HTTP status codes that are returned by ACS.

Table 4 Usage of HTTP Status Codes

Status Code	Message	Usage in ACS	Comment
200	OK	Successful Get, create and query	–
204	OK with no content	Successful delete and update	No data is returned in the response body.
400	Bad Request	Request errors: Object validation failure, XML syntax error, and other error in request message	The request contains bad syntax or cannot be executed. 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.

Table 4 Usage of HTTP Status Codes (continued)

Status Code	Message	Usage in ACS	Comment
401	Unauthorized	Authentication Failure/ Time outs	Similar to 403 error, but specifically for use when authentication failed or credentials are not available.
403	Forbidden	ACS is a secondary and cannot fulfill the request, or operation is not allowed per administrator authorization.	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 sent to a secondary instance.
404	Not Found	For cases where the URL is wrong or the REST service is not enabled	–
410	Gone	A resource is no longer available.	A request was made for an object that does not exist. For example, deleting an object that does not exist.
500	Internal Server Error	For any server error that has no specific HTTP code.	–

ACS REST Result

The HTTP response for a REST request includes either requested objects or a REST result object. See Table 3 on page 7 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:

1. From the ACS user interface, go to **System Administration > Downloads > Rest Service**
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 four XSD files that describe the structure of the objects that are supported on ACS 5.8 REST interfaces.

The four XSD files are:

- Common.xsd, which describes the following objects:
  - Version
  - AttributeInfo
  - Error Message
  - ResultResult, RestCreateResult
  - BaseObject
  - Service Location
  - Status
  - RestCommonOperationType
- Identity.xsd, which describes the following objects:
  - Users
  - Hosts
  - IdentityGroup
- Query.xsd, which describes the structure of query objects.
- NetworkDevice.xsd, which describes the following objects:
  - Network Devices
  - Network Device Groups
  - Network Device Group Types

You can download the schema files in the same way as you download the WADL files. You can use the schema with available tools such as JAXB to generate schema classes.

You can develop HTTP client or use any third-party HTTP client code and integrate it with the schema classes that are generated from the XSD files.

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

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
- Create, Delete, Update Network Device
- Get Network Devices by ID and by name
- Get All Network Devices
- Query for Network Device
- Create, Delete, Update Network Device Group
- Get Network Device Group by ID or by name
- Get sub-tree of Network Device Group
- Query for Network Device Group
- Create, Delete, Update Network Device Group Type
- Get Network Device Group Type by ID and by name
- Get All Network Device Group Type
- Query for Network Device Group Type

- Create, Delete, Update Internal Host
- Get Internal Host by ID or Name
- Get All Internal Host
- Query for Internal Host

Changing Internal User Password from REST API

ACS allows you to change the user password from REST API. You can use the **GET** method from REST API to retrieve the change password XML file from ACS. You can enter the old password and new password in the retrieved XML file and use the **PUT** method to update the password in ACS. This feature is applicable only for the internal users.

Before you Begin:

Ensure that the REST API is enabled in ACS. To enable REST API, execute the **acs config-web-interface rest enable** command in EXEC mode of ACS CLI.

To change user password from REST API:

1. Open the REST API client.
2. Enter **https://<*IP address*>/Rest/UCP/User/name/<*username*>** in the **Request URL** field; where <*IP address*> is the IP address of the ACS instance in which the user account is present and <*username*> is the username of the user for whom you want to change the password.
3. Select **GET** from the Method drop-down list and click **SEND**.

   The REST API displays the Response dialog box with the following XML code:

   ```
   <?xml version"1.0" encoding "UTF-8" standalone "yes"?>
   <ns2:userPassword
   xmlns:ns2-"ucp.rest.mgmt.acs.nm.cisco.com">
   <version>0</version><newPassword>*******</newPassword>
   <oldPassword>*******</oldPassword>
   (userName><acsuser></userName></ns2:userPassword>
   ```

   Observe that the old password and new password fields are displayed as asterisks.
4. Enter the old password and the new password in the XML code.

   For example:

   ```
   <?xml version"1.0" encoding "UTF-8" standalone "yes"?>
   <ns2:userPassword
   xmlns:ns2-"ucp.rest.mgmt.acs.nm.cisco.com">
   <version>0</version><newPassword>123456</newPassword>
   <oldPassword>abcdefg</oldPassword>
   <userName><acsuser></userName></ns2:userPassword>
   ```
5. Change the **Request URL** field to **https://<*IP address*>/Rest/UCP/User**.
6. Select **PUT** from the Method drop-down list and click **SEND**.

   REST API displays the response dialog box with the status as “204 No Content” after successfully changing the password.

   If the password changing operation fails, then REST API displays the corresponding error codes and error messages in the XML code. See Table 4 on page 9 for more information on Error Codes.

Sample Code