Using the Cisco UCS Manager XML API Methods

This chapter includes the following sections:

- Authentication Methods, on page 1
- Query Methods, on page 4
- Using Query Methods for Statistics, on page 8
- Querying Faults, on page 9
- Using Filters, on page 9

Authentication Methods

Authentication allows XML API interaction with the Cisco UCS. It provides a way to set permissions and control the operations that can be performed.

Note Most code examples in this guide substitute the term `<real_cookie>` for an actual cookie (such as 1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf). The Cisco UCS cookie is a 47-character string; it is not the type of cookie that web browsers store locally to maintain session information.

Login

To log in, the XML API client establishes a TCP connection to the Cisco UCS Manager HTTP (or HTTPS) server and posts an XML document containing the `aaaLogin` method.

In the following example, the Telnet utility is used to establish a TCP connection to port 80 of the Cisco UCS Manager with IP address 192.0.20.72. The path used is /nuova.

```
$ telnet 192.0.20.72 80
POST /nuova HTTP/1.1
USER-Agent: lwp-request/2.06
HOST: 192.0.20.72
Content-Length: 62
Content-Type: application/x-www-form-urlencoded
```

Next, the client specifies the `aaaLogin` method and provides a user name and password:

```
<aaaLogin
    inName="admin"
    inPassword="password" />
```

Note Do not include XML version or DOCTYPE lines in the XML API document. The inName and inPassword attributes are parameters.

Each XML API document represents an operation to be performed. When the request is received as an XML API document, Cisco UCS reads the request and performs the actions as provided in the method. Cisco UCS responds with a message in XML document format and indicates success or failure of the request.

The following is a typical successful response:

```
1 <aaaLogin
2    response="yes"
3    outCookie="<real_cookie>"
4       outRefreshPeriod="600"
5       outPriv="aaa,ext-lan-policy,ext-lan-qos,ext-san-policy,operations,
                 pod-policy,pod-qos,read-only"
6    outDomains="mgmt02-dummy"
7    outChannel="noencssl"
8    outEvtChannel="noencssl">
9 </aaaLogin>
```

Each line in the response should be interpreted as follows:

1. Specifies the method used to login.
2. Confirms that this is a response.
3. Provides the session cookie.
4. Specifies the recommended cookie refresh period. The default login session length is 600 seconds.
5. Specifies the privilege level assigned to the user account.
6. The outDomains value is `mgmt02-dummy`.
7. The outChannel value of `noencssl` declares that this session is not using encryption over SSL.
8. The outEvtChannel value of `noencssl` declares that any event subscriptions would not use encryption over SSL.
9. Closing tag.

Alternatively, you can use the cURL utility to log in to the XML API, as shown in the following example:

```
curl -d "<aaaLogin inName='admin' inPassword='password'></aaaLogin>" http://192.0.20.72/nuova
```

If HTTPS is enabled, you must use HTTPS in the cURL command, as shown in the following example:

```
curl -d "<aaaLogin inName='admin' inPassword='password'></aaaLogin>" https://192.0.20.72/nuova
```

Refreshing the Session

Sessions are refreshed with the `aaaRefresh` method, using the 47-character cookie obtained either from the `aaaLogin` response or a previous refresh.

```
<aaaRefresh
     inName="admin"
     inPassword="mypasword"
     inCookie="real_cookie"/>
```

Logging Out of the Session

Use the following method to log out of a session:

```
<aaaLogout
     inCookie="<real_cookie>" />
```

Unsuccessful Responses

Failed login:

```
<aaaLogin
    response="yes"
    cookie="<real_cookie>"
    errorCode="551"
    invocationResult="unidentified-fail"
    errorDescr="Authentication failed">
</aaaLogin>
```

Nonexistent object (blank return indicates no object with the specified DN):

```
<configResolveDn
    dn="sys-machine/chassis-1/blade-4711"
    cookie="<real_cookie>"
    response="yes">
    <outConfig> </outConfig>
</configResolveDn>
```

Bad request:

```
<configConfMo
    dn="fabric/server"
    cookie="<real_cookie>"
    response="yes"
    errorCode="103"
    invocationResult="unidentified-fail"
    errorDescr="can't create; object already exists.">
</configConfMo>
```

Query Methods

Using `configFindDnsByClassId`

When finding distinguished names of a specified class, note the following:

- This method retrieves the DNs of a specified class.
- classId specifies the object type to retrieve (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

See the example request/response in configFindDnsByClassId.

Using `configResolveChildren`

When resolving children of objects in the MIT, note the following:

- This method obtains all child objects of a named object that are instances of the named class. If a class name is omitted, all child objects of the named object are returned.
- inDn attribute specifies the named object from which the child objects are retrieved (required).
- classId attribute specifies the name of the child object class to return (optional).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

See the example request/response in configResolveChildren.

Using `configResolveClass`

When resolving a class, note the following:

- All objects of the specified class type are retrieved.
- classId specifies the object class name to return (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

Result sets can be large. Be precise when defining result sets. For example, to obtain only a list of servers, use **`computeItem`** as the attribute value for classId in the query. To get all instances of equipment, query the `equipmentItem` class. This example queries for all instances of the `equipmentItem` class:

```
<configResolveClass
 cookie="real_cookie"
 inHierarchical="false"
 classId="equipmentItem"/>
```

See the example request/response in configResolveClass.

Using `configResolveClasses`

When resolving multiple classes, note the following:

- This method retrieves all the objects of the specified class types.
- classId attribute specifies the name of the object class to return (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

If an invalid class name is specified in the inId attribute, an XML parsing error is generated and the query cannot execute.

See the example request/response in configResolveClasses.

Using `configResolveDn`

When resolving a DN, note the following:

- The object specified by the DN is retrieved.
- Specified DN identifies the object instance to be resolved (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

See the example request/response in configResolveDn.

Using `configResolveDns`

When resolving multiple DNs, note the following:

- The objects specified by the DNs are retrieved.
- Specified DN identifies the object instance to be resolved (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

- Order of a request does not determine the order of the response.
- Unknown DNs are returned as part of the `outUnresolved` element.

See the example request/response in configResolveDns.

Using `configResolveParent`

When resolving the parent object of an object, note the following:

- This method retrieves the parent object of a specified DN.
- dn attribute is the DN of the child object (required).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

See the example request/response in configResolveParent.

Using `configScope`

Limiting the scope of a query allows for a finer grained, less resource-intensive request. The query can be anchored at a point in the management information tree other than the root. When setting the query scope, note the following:

- This method sets the root (scope) of the query to a specified DN and returns objects of the specified class type.
- dn is the named object from which the query is scoped (required).
- inClass attribute specifies the name of the object class to return (optional; when a class is not specified, the query acts the same as configResolveDn).
- Authentication cookie (from `aaaLogin` or `aaaRefresh`) is required.
- inHierarchical attribute (default = false) if true, specifies that results are hierarchical.
- Enumerated values, `classIds`, and bit masks are displayed as strings.

The following example is a query for the Ethernet interfaces on the blades in chassis 1:

```
<configScope
    dn="sys/chassis-1"
    inClass="adaptorExtEthIf"
    cookie="<real_cookie>"
    inHierarchical="false"/>
```

Also see the example request/response in configScope.

Querying the MAC Pool

To obtain a list of all MAC addresses, query for **`macpoolAddr`**. These are children of the (system-created) `macpoolUniverse`. The request is as follows:

```
<configScope
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="mac" inClass="macpoolAddr"/>
```

The response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<configScope
    cookie="<real_cookie>"
    dn="mac" response="yes">
    <outConfigs>
       <macpoolAddr
          assigned="no"
          assignedToDn=""
          dn="mac/00:00:00:00:FF:0F"
          id="00:00:00:00:FF:0F"
          owner="pool">
       </macpoolAddr>
       <macpoolAddr
          assigned="no"
          assignedToDn=""
          dn="mac/00:00:00:00:FF:0E"
          id="00:00:00:00:FF:0E"
          owner="pool">
       </macpoolAddr>
       .
       .
       .
    <\outconfig
<\configScope
```

Because the objects of the `macpoolAddr` class can exist only as children of the MAC pool universe, a simpler query follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="macpoolAddr"/>
```

To determine which `computeItem` (blade or rack mount server) is assigned a particular MAC address, specify the MAC address in the query and look at the `assignedToDn` field in the response. For example, a request with a specified MAC address follows:

```
<configResolveDn
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="mac/10:00:00:00:00:03"/>
```

The response is as follows:

```
<configResolveDn
    dn="mac/10:00:00:00:00:03"
    cookie="real_cookie"
    response="yes">
    <outConfig>
       <macpoolAddr assigned="yes"
          assignedToDn="org-root/ls-BOB/ether-eth1"
          dn="mac/10:00:00:00:00:03" id="10:00:00:00:00:03"
          owner="pool" />
    </outConfig>
</configResolveDn>
```

Using Query Methods for Statistics

Statistics are available on a wide range of objects. Querying all statistics at once is resource intensive. Instead, identify the type of statistic and the object on which it reports; for example, getting the `compCpuStats` object for `sys/chassis-1/blade-1/board/cpu-2`, query for all children of `sys/chassis-1/blade-1/board/cpu-2`. The request is as follows:

```
<configScope
    cookie="real_cookie"
    inHierarchical="false"
    dn="sys/chassis-1/blade-1/board/cpu-2"
    inClass="processorEnvStats" />
```

The system response is as follows:

```
<configScope
    dn="sys/chassis-1/blade-5/board/cpu-2"
    cookie="<real_cookie>"
    response="yes">
    <outConfigs>
       <processorEnvStats
          dn="sys/chassis-1/blade-5/board/cpu-2/env-stats"
          inputVoltage="0.058200"
          inputVoltageAvg="0.062080"
          inputVoltageMax="0.077600"
          inputVoltageMin="0.058200"
          intervals="58982460"
          suspect="no"
          temperature="23.000000"
          temperatureAvg="23.000000"
          temperatureMax="23.000000"
          temperatureMin="23.000000"
          thresholded=""
          timeCollected="2009-09-23T12:40:55"
          update="327680"/>
    </outConfigs>
</configScope>
```

A query by the DN is more efficient:

```
<configResolveDn
    inHierarchical="false"
    cookie="<real_cookie>"
```

```
    dn="sys/chassis-1/blade-1/board/cpu-2/processorEnvStats">
</configResolveDn>
```

With the statistic object's DN, query for historical statistics on the children of the object using a hierarchical query. To get the statistic object and historical statistics, change inHierarchical to true:

```
<configResolveDn
    inHierarchical="true"
    cookie="<real_cookie>"
    dn="sys/chassis-1/blade-1/board/cpu-2/processorEnvStats">
</configResolveDn>
```

Querying Faults

The following example obtains a list of major faults:

```
<configResolveClass
    cookie="real_cookie"
    inHierarchical="false"
    classId="faultInst"/>
```

The following example (which contains the filter `<inFilter>eq class= “faultInst”`) obtains a list of all major or critical faults:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="faultInst">
    <inFilter>
       <eq class="faultInst"
          property="highestSeverity"
          value="major" />
    </inFilter>
</configResolveClass>
```

Using Filters

Simple Filters

Simple filters use true and false conditions of properties to select results.

False example:

```
<configResolveClass
    cookie="<real_cookie>"
    classId="topSystem"
    inHierarchical="false">
    <inFilter>
    </inFilter>
```

```
</configResolveClass>
```

True example:

```
<configResolveClass
    cookie="<real_cookie>"
    classId="topSystem"
    inHierarchical="true">
    <inFilter>
    </inFilter>
</configResolveClass>
```

Property Filters

Equality Filter

The example shows a query for all associated servers. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="lsServer">
    <inFilter>
       <eq class="lsServer"
          property="assocState"
          value="associated" />
    </inFilter>
</configResolveClass>
```

Not Equal Filter

The example finds all unassigned servers (assignment state property is not equal to **assigned**). The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="lsServer">
    <inFilter>
       <ne class="lsServer"
          property="assignState"
          value="assigned" />
    </inFilter>
</configResolveClass>
```

Greater Than Filter

The example finds the memory arrays with more than 1024 MB capacity. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="memoryArray">
    <inFilter>
       <gt class="memoryArray"
          property="currCapacity"
```

```
            value="1024" />
    </inFilter>
</configResolveClass>
```

Greater Than or Equal to Filter

The example finds the memory arrays with 2048 MB capacity or more. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="memoryArray">
    <inFilter>
       <ge class="memoryArray"
          property="currCapacity"
          value="2048" />
    </inFilter>
</configResolveClass>
```

Less Than Filter

The example finds memory arrays with less than 1024 MB capacity. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="memoryArray">
    <inFilter>
       <lt class="memoryArray"
          property="currCapacity"
          value="1024" />
    </inFilter>
</configResolveClass>
```

Less Than or Equal to Filter

The example finds memory arrays with 2048 MB capacity or less. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="memoryArray">
    <inFilter>
       <le class="memoryArray"
          property="currCapacity"
          value="2048" />
    </inFilter>
</configResolveClass>
```

Wildcard Filter

The wildcard filter uses standard regular expression syntax. The example finds any adapter unit whose serial number begins with the prefix **QCI1**:

```
<configResolveClass
    cookie="<real_cookie>"
```

```
    inHierarchical="false"
    classId="adaptorUnit">
    <inFilter>
       <wcard class="adaptorUnit"
          property="serial"
          value="QCI1.*" />
    </inFilter>
</configResolveClass>
```

Any Bits Filter

This example finds all servers that have a `connStatus` of either **A** or **B** (the property `connStatus` is a bit mask). The filter is framed as follows:

```
<configResolveClass
    cookie="null"
    inHierarchical="false"
    classId="computeItem">
    <inFilter>
       <anybit class="computeItem"
          property="connStatus"
          value="A,B" />
    </inFilter>
</configResolveClass>
```

All Bits Filter

The example finds all service profiles with the `configQualifier` bit mask set to both **vnic-capacity** and **vhba-capacity**. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="lsServer">
    <inFilter>
       <allbits class="lsServer"
          property="configQualifier"
          value="vnic-capacity,vhba-capacity" />
    </inFilter>
</configResolveClass>
```

Composite Filters

AND Filter

To determine all UUIDs assigned and owned by pools, run the following query. The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="uuidpoolAddr">
    <inFilter>
       <and>
          <eq class="uuidpoolAddr"
             property="owner"
             value="pool" />
          <eq class="uuidpoolAddr"
```

```
            property="assigned"
            value="yes" />
      </and>
   </inFilter>
</configResolveClass>
```

The response is as follows:

```
<configResolveClass
    classId="uuidpoolAddr"
    cookie="<real_cookie>"
    response="yes">
    <outConfigs>
       <uuidpoolAddr
          assigned="yes"
          assignedToDn="org-root/ls-foo"
          dn="uuid/F000-00000000000F"
          id="F000-00000000000F"
          owner="pool">
       </uuidpoolAddr>
    </outConfigs>
</configResolveClass>
```

In the example, the AND filter finds the chassis with vendor **`Cisco Systems Inc`** and serial number **`CHS A04`**:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="equipmentChassis">
    <inFilter>
       <and>
          <eq property="vendor"
             value="Cisco Systems Inc"
             class="equipmentChassis"/>
          <eq class="equipmentChassis"
             property="serial"
             value="CHS A04"/>
       </and>
    </inFilter>
</configResolveClass>
```

OR Filter

The example returns all objects of type `computeItem` that are located in slot one or slot eight from all chassis.

```
<configResolveClass
    inHierarchical="false"
    cookie="<real_cookie>"
    classId="compute">
    <inFilter>
       <or>
          <eq class="computeItem"
             property="slotId"
             value="1"/>
          <eq class="computeItem"
             property="slotId"
             value="8"/>
        </or>
```

```
    </inFilter>
</configResolveClass>
```

Between Filter

The example finds the memory arrays with slots 1, 2, 3, 4, or 5 populated (note that the between range is inclusive). The filter is framed as follows:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="memoryarray">
    <inFilter>
       <bw class="memoryArray"
          property="populated"
          firstValue="1"
          secondValue="5"/>
    </inFilter>
</configResolveClass>
```

AND, OR, NOT Composite Filter

The example is an AND, OR, NOT combination. It returns all objects of the `computeItem` type that are located in slot one or slot eight from all chassis, except chassis five.

```
<configResolveClass
    inHierarchical="false"
    cookie="<real_cookie>"
    classId="computeItem">
    <inFilter>
       <and>
          <or>
             <eq class="computeItem" property="slotId" value="1"/>
             <eq class="computeItem" property="slotId" value="8"/>
          </or>
          <not>
             <eq class="computeItem" property="chassisId" value="5"/>
          </not>
       </and>
    </inFilter>
</configResolveClass>
```

NOT Modifier Filter

The NOT filter can negate a contained filter. The filter is framed as follows. The example queries for servers that do not have a `connStatus` of unknown (the property `connStatus` is a bit mask).

```
<configResolveClass
    cookie="null"
    inHierarchical="false"
    classId="computeItem">
    <inFilter>
       <not>
          <anybit class="computeItem"
             property="connStatus"
             value="unknown" />
       </not>
```

```
    </inFilter>
</configResolveClass>
```