Cisco UCS Manager XML API

This chapter includes the following sections:

- Information About the Cisco UCS Manager XML API, page 1
- Cisco Unified Computing System Overview, page 2
- Cisco UCS Management Information Model, page 2
- Cisco UCS Manager XML API Sample Flow, page 3
- Object Naming, page 5
- API Method Categories, page 5
- Success or Failure Response, page 10

Information About the Cisco UCS Manager XML API

The Cisco UCS Manager XML API is a programmatic interface to Cisco Unified Computing System (UCS). The API accepts XML documents through HTTP or HTTPS. Developers can use any programming language to generate XML documents that contain the API methods. Configuration and state information for Cisco UCS is stored in a hierarchical tree structure known as the management information tree, which is completely accessible through the XML API.

The Cisco UCS Manager XML API supports operations on a single object or an object hierarchy. An API call can initiate changes to attributes of one or more objects such as chassis, blades, adapters, policies, and other configurable components.

The API operates in forgiving mode. Missing attributes are replaced with applicable default values that are maintained in the internal data management engine (DME). The DME ignores incorrect attributes. When multiple managed objects (MOs) are being configured, the API operation stops if any of the MOs (a virtual NIC, for example) cannot be configured. In that case, the management information tree is rolled back to the prior state that preceded the API operation and an error is returned.

Updates to MOs and properties conform to the existing object model to ensure backward compatibility. If existing properties are changed during a product upgrade, they are managed during the database load after the upgrade. New properties are assigned default values.

Operation of the API is transactional and terminates on a single data model. Cisco UCS is responsible for all endpoint communication, such as state updates. Users cannot communicate directly to endpoints, which relieves developers from administering isolated, individual component configurations.

The API model includes the following programmatic entities:

- Classes—Define the properties and states of objects in the management information tree.
- Methods—Actions that the API performs on one or more objects.
- Types—Object properties that map values to the object state (for example, `equipmentPresence`).

A typical request comes into the DME and is placed in the transactor queue in FIFO order. The transactor gets the request from the queue, interprets the request, and performs an authorization check. After the request is confirmed, the transactor updates the management information tree. This complete operation is done in a single transaction.

Full event subscription is enabled. After subscribing, any event notification is sent along with its type of state change.

Cisco Unified Computing System Overview

A Cisco UCS domain can consist of up to two Cisco UCS fabric interconnects and a minimum of one Cisco chassis with one blade or rack-mounted server. Up to 40 chassis with a mixture of blade and rack-mounted servers can be connected and controlled by a single Cisco UCS domain.

Cisco UCS Manager runs on the primary fabric interconnect, with failover capability to the subordinate fabric interconnect. In the event of a failover, the virtual IP address will connect to the subordinate fabric interconnect, making it the new primary fabric interconnect.

All XML requests to Cisco UCS are asynchronous and terminate on the active Cisco UCS Manager. Cisco UCS Manager mediates all communication within the system; no direct user access to the Cisco UCS components is required.

Cisco UCS Manager is aware of the current configuration and performs automated device discovery whenever a new resource is installed. After a resource is detected, Cisco UCS Manager adds it and its characteristics to the system inventory. Cisco UCS Manager can preconfigure the new resources if it is directed to do so by an administrator-defined policy.

Cisco UCS Management Information Model

All the physical and logical components that comprise Cisco UCS are represented in a hierarchical management information model (MIM), also referred to as the MIT. Each node in the tree represents a managed object (MO) or group of objects that contains its administrative state and its operational state.

The hierarchical structure starts at the top (`sys`) and contains parent and child nodes. Each node in this tree is a managed object and each object in Cisco UCS has a unique distinguished name (DN) that describes the object and its place in the tree. Managed objects are abstractions of the Cisco UCS resources, such as fabric interconnects, chassis, blades, and rack-mounted servers.

Configuration policies are the majority of the policies in the system and describe the configurations of different Cisco UCS components. Policies determine how the system behaves under specific circumstances. Certain managed objects are not created by users, but are automatically created by the Cisco UCS, for example, power supply objects and fan objects. By invoking the API, you are reading and writing objects to the MIM.

The information model is centrally stored and managed by the data management engine (DME), a user-level process running on the fabric interconnects. When a user initiates an administrative change to a Cisco UCS component (for example, applying a service profile to a server), the DME first applies that change to the information model, and then applies the change to the actual managed endpoint. This approach is called a model-driven framework.

The following is a branch diagram that starts at `sys` from the `topRoot` of the Cisco UCS management information tree. The diagram shows a hierarchy that consists of five populated chassis with eight blades in each chassis. All the blades shown have one or more adapters. For simplicity, only chassis number five is expanded.

Figure 1: Illustration of MIM Structure Showing Five Chassis

```
Tree (topRoot):—————————————————————————-Distinguished Name:

|———sys————————————————————— (sys)
     |———chassis-1——————————————(sys/chassis-1)
     |———chassis-2——————————————(sys/chassis-2)
     |———chassis-3——————————————(sys/chassis-3)
     |———chassis-4——————————————(sys/chassis-4)
     |———chassis-5——————————————(sys/chassis-5)
          |———blade-1————————— (sys/chassis-5/blade-1)
               |———adaptor-1——— (sys/chassis-5/blade-1/adaptor-1)
          |———blade-2————————— (sys/chassis-5/blade-2)
               |———adaptor-1——— (sys/chassis-5/blade-2/adaptor-1)
               |———adaptor-2——— (sys/chassis-5/blade-2/adaptor-2)
          |———blade-3————————— (sys/chassis-5/blade-3)
               |———adaptor-1——— (sys/chassis-5/blade-3/adaptor-1)
               |———adaptor-2——— (sys/chassis-5/blade-3/adaptor-2)
          |———blade-4————————— (sys/chassis-5/blade-4)
               |———adaptor-1——— (sys/chassis-5/blade-4/adaptor-1)
          |———blade-5————————— (sys/chassis-5/blade-5)
               |———adaptor-1——— (sys/chassis-5/blade-5/adaptor-1)
               |———adaptor-2——— (sys/chassis-5/blade-5/adaptor-2)
          |———blade-6————————— (sys/chassis-5/blade-6)
               |———adaptor-1——— (sys/chassis-5/blade-6/adaptor-1)
          |———blade-7————————— (sys/chassis-5/blade-7)
               |———adaptor-1——— (sys/chassis-5/blade-7/adaptor-1)
          |———blade-8————————— (sys/chassis-5/blade-8)
               |———adaptor-1——— (sys/chassis-5/blade-8/adaptor-1)
```

Cisco UCS Manager XML API Sample Flow

A typical request comes into the data management engine (DME) and is placed in the transactor queue in FIFO order. The transactor gets the request from the queue, interprets the request, and performs an authorization check. After the request is confirmed, the transactor updates the management information tree. This operation is done in a single transaction.

The following figure shows how Cisco UCS Manager processes a boot server request. The following table describes the steps involved in a boot server request.

Figure 2: Sample Flow of Boot Server Request

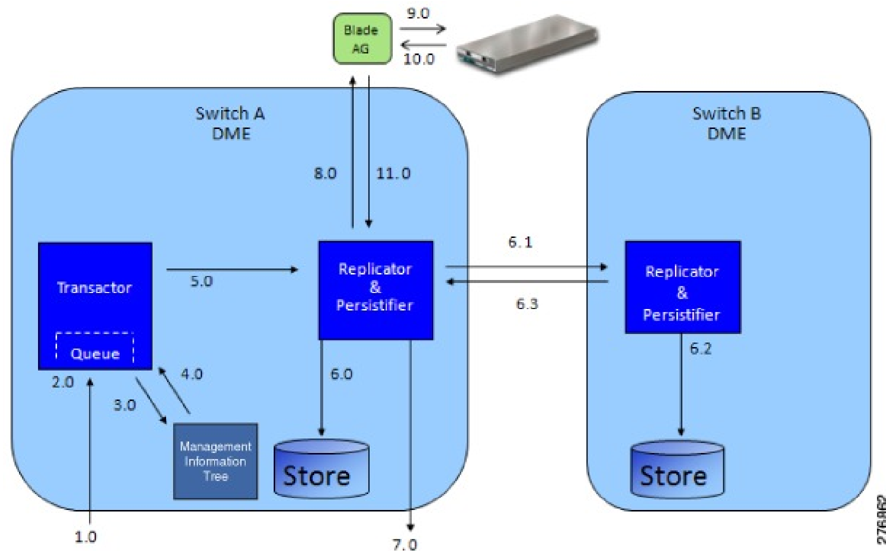

Table 1: Explanation of Boot Server Request

Step	Command/Process	Administrative Power State of MO (Server)	Operational Power State of MO (Server)
1	CMD request: boot server	Down	Down
2	Request queued	Down	Down
3	State change in management information tree	Up	Down
4	Transaction complete	Up	Down
5	Pass change information and boot request stimuli	Up	Down
6.0	Make persistent the managed object (MO) state change	Up	Down
6.1	Send state change information to peer DME	Up	Down
6.2	Make persistent the MO state to peer’s local store	Up	Down
6.3	Reply with success (replication and persistence)	Up	Down
7	CMD: response and external notification	Up	Down
8	Apply boot stimuli	Up	Down

Step	Command/Process	Administrative Power State of MO (Server)	Operational Power State of MO (Server)
9	Instruct BMC to power on server	Up	Down
10	Reply from BMC: server power on success	Up	Up
11	Reply, boot stimuli success, pass new power state information	Up	Up

Object Naming

You can identify a specific object by its distinguished name (DN) or by its relative name (RN).

Distinguished Name

The distinguished name enables you to unambiguously identify a target object. The distinguished name has the following format consisting of a series of relative names:

```
dn = {rn}/{rn}/{rn}/{rn}...
```

In the following example, the DN provides a fully qualified path for `adaptor-1` from the top of the object tree to the object. The DN specifies the exact managed object on which the API call is operating.

```
< dn =”sys/chassis-5/blade-2/adaptor-1” />
```

Relative Name

The relative name identifies an object within the context of its parent object. The distinguished name is composed of a sequence of relative names.

For example, this distinguished name:

```
<dn = "sys/chassis-5/blade-2/adaptor-1/host-eth-2"/>
```

is composed of the following relative names:

```
topSystem MO: rn="sys"
equipmentChassis MO: rn="chassis-<id>"
computeBlade MO: rn ="blade-<slotId>"
adaptorUnit MO: rn="adaptor-<id>"
adaptorHostEthIf MO: rn="host-eth-<id>"
```

API Method Categories

Each method corresponds to an XML document.

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

Authentication Methods

Authentication methods authenticate and maintain the session. For example:

- aaaLogin—Initial method for logging in.
- aaaRefresh—Refreshes the current authentication cookie.
- aaaLogout—Exits the current session and deactivates the corresponding authentication cookie.

Use the aaaLogin method to get a valid cookie. Use aaaRefresh to maintain the session and keep the cookie active. Use the aaaLogout method to terminate the session (also invalidates the cookie). A maximum of 256 sessions to the Cisco UCS can be opened at any one time.

Operations are performed using the HTTP post method (Cisco UCS supports both HTTP and HTTPS requests) over TCP. HTTP and HTTPS can be configured to use different port numbers, but TCP/443 (or TCP/80 for non-secure connections) is used by default. The HTTP envelope contains the XML configuration.

Tip In CIMC, HTTP to HTTPS redirection is enabled by default. To capture HTTP packets between the client application and CIMC, disable redirection in the CIMC GUI or CLI.

Query Methods

Query methods obtain information on the current configuration state of an object. The following are query examples:

- configResolveDn—Retrieves objects by DN.
- configResolveDns—Retrieves objects by a set of DNs.
- configResolveClass—Retrieves objects of a given class.
- configResolveClasses—Retrieves objects of multiple classes.
- configFindDnsByClassId—Retrieves the DNs of a specified class.
- configResolveChildren—Retrieves the child objects of an object.
- configResolveParent—Retrieves the parent object of an object.
- configScope—Performs class queries on a DN in the management information tree.

Most query methods have the argument inHierarchical (Boolean true/yes or false/no). If true, the inHierarchical argument returns all child objects.

```
<configResolveDn … inHierarchical="false"></>
<configResolveDn … inHierarchical="true"></>
```

Because the amount of data returned from Cisco UCS can be quite large, the inHierarchical argument should be used with care. For example, if the query method is used on a class or DN that refers to a managed object (MO) that is located high on the management information tree and inHierarchical is set to true, the response can contain almost the entire Cisco UCS configuration. The resources required for Cisco UCS to process the request can be high, causing Cisco UCS to take an extended amount of time to respond. To avoid delays, the query method should be performed on a smaller scale involving fewer MOs.

Tip If a query method does not respond or is taking a long time to respond, increase the timeout period on the client application or adjust the query method to involve fewer MOs.

The query API methods might also have an inRecursive argument to specify whether the call should be recursive (for example, follow objects that point back to other objects or the parent object).

The API also provides a set of filters to increase the usefulness of the query methods. These filters can be passed as part of a query and are used to identify the wanted result set.

Note Until a host is powered on at least once, Cisco UCS may not have complete inventory and status information. For example, if Cisco UCS is reset, it will not have detailed CPU, memory, or adapter inventory information until the next time the host is powered on. If a query method is performed on a MO corresponding to the unavailable data, the response will be blank.

Simple Filters

There are two simple filters, the true filter and false filter. These two filters react to the simple states of true or false, respectively.

- True filter—Result set of objects with the Boolean condition of true.
- False filter—Result set of objects with the Boolean condition of false.

Property Filters

The property filters use the values of an object's properties as the criteria for inclusion in a result set. To create most property filters, `classId` and `propertyId` of the target object/property is required, along with a value for comparison.

- Equality filter—Restricts the result set to objects with the identified property of "equal" to the provided property value.
- Not equal filter—Restricts the result set to objects with the identified property of "not equal" to the provided property value.

- Greater than filter—Restricts the result set to objects with the identified property of "greater than" the provided property value.
- Greater than or equal filter—Restricts the result set to objects with the identified property of "is greater than or equal" to the provided property value.
- Less than filter—Restricts the result set to objects with the identified property of "less than" the provided property value.
- Less than or equal filter—Restricts the result set to objects with the identified property of "less than or equal" to the provided property value.
- Wildcard filter—Restricts the result set to objects with the identified property matches that includes a wildcard. Supported wildcards include "%" or "*" (any sequence of characters), "?" or "-" (any single character).
- Any bits filter—Restricts the result set to objects with the identified property that has at least one of the passed bits set. (Use only on bitmask properties.)
- All bits filter—Restricts the result set to objects with the identified property that has all the passed bits set. (Use only on bitmask properties.)

Composite Filters

The composite filters are composed of two or more component filters. They enable greater flexibility in creating result sets. For example, a composite filter could restrict the result set to only those objects that were accepted by at least one of the contained filters.

- AND filter—Result set must pass the filtering criteria of each component filter. For example, to obtain all compute blades with `totalMemory` greater than 64 megabytes and operability of operable, the filter is composed of one greater than filter and one equality filter.
- OR filter—Result set must pass the filtering criteria of at least one of the component filters. For example, to obtain all the service profiles that have an `assignmentState` of unassigned or an association state value of unassociated, the filter is composed of two equality filters.
- Between filter—Result set is those objects that fall between the range of the first specified value and second specified value, inclusive. For example, all faults that occurred starting on the first date and ending on the last date.
- XOR filter—Result set is those objects that pass the filtering criteria of no more than one of the composite's component filters.

Modifier Filter

A modifier filter changes the results of a contained filter.

The only modifier filter that is currently supported is the NOT filter that negates the result of a contained filter. Use this filter to obtain objects that do not match contained criteria.

Configuration Methods

There are several methods to make configuration changes to managed objects. These changes can be applied to the whole tree, a subtree, or an individual object. The following are examples of configuration methods:

- configConfMo—Affects a single managed object (for example, a DN).
- configConfMos—Affects multiple subtrees (for example, several DNs).
- configConfMoGroup—Makes the same configuration changes to multiple subtree structures (DNs) or managed objects.

Most configuration methods use the argument inHierarchical (Boolean true/yes or false/no). These values do not play a significant role during configuration because child objects are included in the XML document and the DME operates in the forgiving mode.

Event Subscription Methods

Applications get state change information by regular polling or event subscription. For more efficient use of resources, event subscription is the preferred method of notification. Polling should be used only under very limited circumstances.

Use eventSubscribe to register for events, as shown the following example:

```
<eventSubscribe
    cookie="<real_cookie>">
</eventSubscribe>
```

To receive notifications, open an HTTP or HTTPS session over TCP and keep the session open. On receiving eventSubscribe, Cisco UCS starts sending all new events as they occur. You can unsubscribe from these events using the eventUnsubscribe method.

Each event has a unique event ID. Event IDs operate as counters and are included in all method responses. When an event is generated, the event ID counter increments and is assigned as the new event ID. This sequential numbering enables tracking of events and ensures that no event is missed.

An event channel connection opened by a user will be closed automatically by Cisco UCS after 600 seconds of inactivity associated with the event channel session cookie. To prevent automatic closing of the event channel connection by Cisco UCS, the user must either send the aaaKeepAlive request for the same event channel session cookie within 600 seconds or send any other XML API method to Cisco UCS using the same event channel session cookie.

Capturing XML Interchange Between the GUI and Cisco UCS

Interchange is stored in a log file such as `C:\Documents and Settings\username\Application Data\Sun\Java\Deployment\log\.ucsm`. Due to internal security requirements, this information is not always complete. However, you can use a commercial packet analyzer application to observe sent XML.

Success or Failure Response

When Cisco UCS responds to an XML API request, the response indicates failure if the request is impossible to complete. A successful response indicates only that the request is valid, not that the operation is complete. For example, it may take some time for a server to finish a power-on request. The power state changes from down to up only after the server actually powers on.

Successful Response

When a request has executed successfully, Cisco UCS returns an XML document with the information requested or a confirmation that the changes were made. The following is an example of a configResolveDn query on the distinguished name `sys/chassis-1/blade-1`:

```
<configResolveDn
     dn="sys/chassis-1/blade-1"
     cookie="<real_cookie>"
     inHierarchical="false"/>
```

The response includes the following information:

```
<configResolveDn dn="sys/chassis-1/blade-1"
     cookie="<real_cookie>"
     response="yes">
     <outConfig>
          <computeItem adminPower="policy"
          adminState="in-service"
          assignedToDn=""
          association="none"
          availability="available"
          chassisId="1"
          checkPoint="discovered"
          connPath="A"
          connStatus="A"
          discovery="complete"
          dn="sys/chassis-1/blade-1"
          fltAggr="0"
          fsmDescr=""
          fsmFlags=""
          fsmPrev="DiscoverSuccess"
          fsmRmtInvErrCode="unspecified"
          fsmRmtInvErrDescr=""
          fsmRmtInvRslt=""
          fsmStageDescr=""
          fsmStamp="2008-11-24T01:27:10"
          fsmStatus="nop"
          fsmTry="0"
          lc="discovered"
          managingInst="A"
          model="Gooding"
          name=""
          numOfAdaptors="1"
          numOfCores="4"
          numOfEthHostIfs="2"
          numOfFcHostIfs="2"
          numOfThreads="0"
          operPower="off"
          operState="unassociated"
          operability="operable"
          originalUuid="1b4e28ba-2fa1-11d2-0101-b9a761bde3fb"
          presence="equipped"
          revision=""
          serial="1-1"
```

```
            slotId="1"
            totalMemory="4096"
            uuid=""
            vendor="Cisco"/>
     </outConfig>
</configResolveDn>
```

Failed Requests

The response to a failed request includes XML attributes for errorCode and errorDescr. The following is an example of a response to a failed request:

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

Empty Results

A query request for a nonexistent object is not treated as a failure by the DME. If the object does not exist, Cisco UCS returns a success message, but the XML document contains an empty data field (`<outConfig> </outConfig>`) to indicate that the requested object was not found. The following example shows the response to an attempt to resolve the distinguished name on a nonexistent rack-mount server:

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