CIMC XML API Programmer’s Guide for Cisco UCS E-Series Servers and the Cisco UCS E-Series Network Compute Engine

First Published: July 15, 2013

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Preface

This preface includes the following sections:

Audience

This guide is intended for software engineers with a background in programming and the use of APIs. Engineers should have knowledge of XML, data systems, networking protocols, and storage protocols.

Document Organization

This XML API Reference Guide is organized into the following chapters and appendixes:

- Cisco CIMC XML API
- Using the Cisco CIMC XML API Methods
- Using CIMC XML API Method Descriptions
- Cisco CIMC XML Object-Access Privileges
- Examples of Common Server Management Tasks
- Notes on Using the configConfMo Method
- The CIMC Visore Utility

Related Documentation

The Documentation Guide for Cisco UCS E-Series Servers and the Cisco UCS E-Series Network Compute Engine provides links to all product documentation.

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, send an email to ucse_docfeedback@cisco.com. We appreciate your feedback.

CHAPTER 1

Cisco CIMC XML API

This chapter includes the following sections:

About the Cisco CIMC XML API

The Cisco Integrated Management Controller (CIMC) XML API is a programmatic interface to the CIMC for the Cisco UCS E-Series Servers (E-Series Servers) and Cisco UCS E-Series Network Compute Engine (NCE). 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 CIMC is stored in a hierarchical tree structure known as the management information tree, which is completely accessible through the XML API.

The Cisco CIMC XML API implements a subset of the methods and management information model available in the Cisco UCS Manager XML API. The behavior of both APIs is similar in syntax and semantics, and you can use the same client development tools and techniques for both. The scope of the Cisco CIMC XML API is limited to a single E-Series Server or NCE, in contrast to the Cisco UCS Manager XML API, which controls an entire UCS environment consisting of switches, FEX modules, servers, and other devices.

Using the Cisco CIMC XML API, the user has programmatic access to CIMC to configure, administer, and monitor the server. The API provides the same functions that are accessible through the CIMC CLI and GUI interfaces.

Operation of the API is transactional and terminates on a single data model maintained in CIMC.

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 CIMC 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.

Event subscription is supported. Up to four Cisco CIMC XML API clients can subscribe to receive event notifications from CIMC. The event subscription operation establishes a connection session allowing a client to receive XML-formatted event notification messages that are sent asynchronously by CIMC.

Note The Cisco CIMC XML API sends event notifications only for fault-related events.

E-Series Servers and the NCE Management Information Model

All the physical and logical components that comprise the E-Series Servers or the NCE are represented in a hierarchical management information model, referred to as the management information tree. 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 CPUs, DIMMs, adapter cards, fans, and power supply units.

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 Cisco UCS, for example, power supply objects and fan objects. By invoking the API, you are reading and writing objects to the management information model (MIM).

CIMC Management Information Model

The CIMC management information model is a subset of the Cisco UCS management information model. An E-Series Server or NCE is modeled starting with `sys/rack-unit-1` in the management information tree as in the following example:

```
|——sys———————————————————— (sys)
     |——rack-unit-1————————————(sys/rack-unit-1)
          |——adaptor-1————————————(sys/rack-unit-1/adaptor-1)
          |——indicator-led-1————————————(sys/rack-unit-1/indicator-led-1)
          |——indicator-led-2————————————(sys/rack-unit-1/indicator-led-2)
```

Cisco CIMC XML API Sample Flow

A typical request comes into CIMC 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 CIMC processes a boot server request, and the following table describes the steps involved in a boot server request.

Figure 1: Sample Flow of Boot Server Request

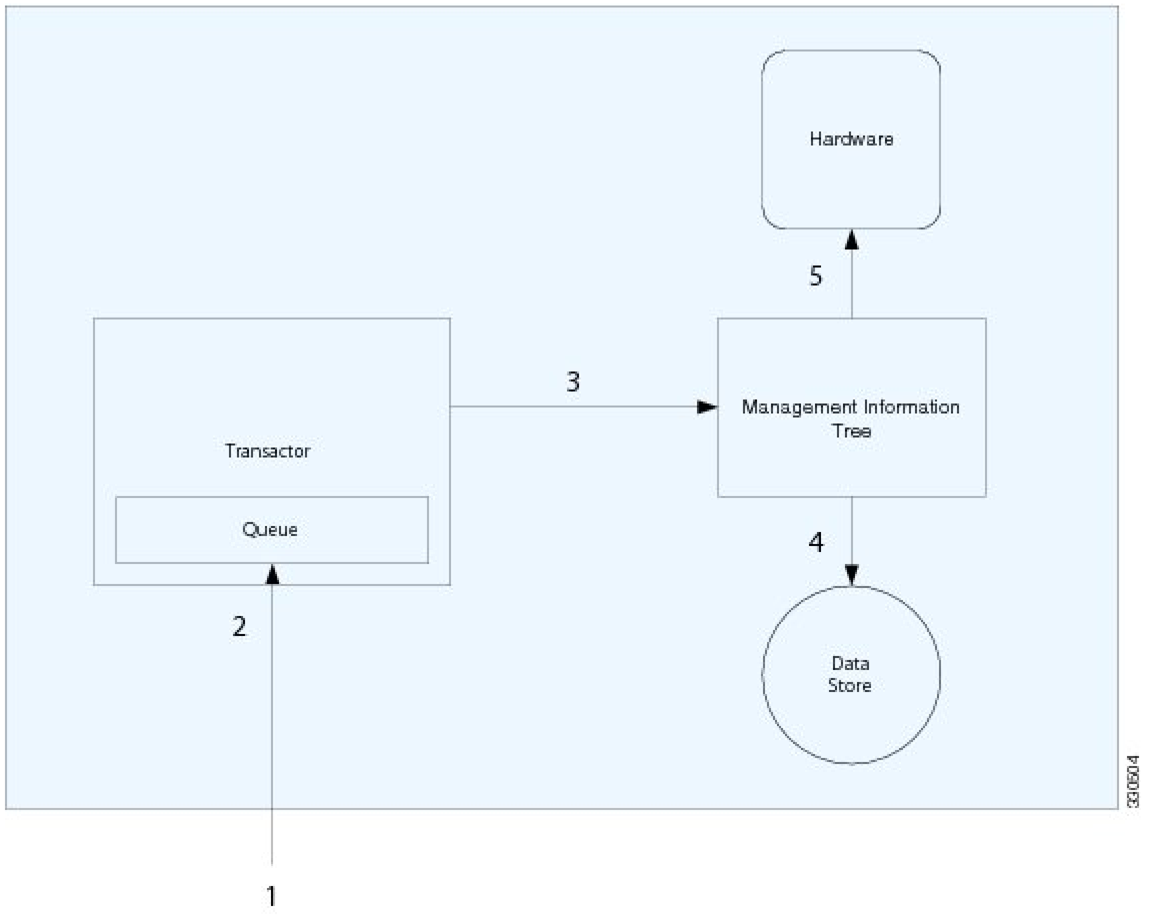

Table 1: Explanation of Boot Server Request

Step	Command/Process	Operational Power State of MO (Server)
1	CMD request: boot server	Down
2	Request queued	Down
3	State change in management information tree	Down
4	Make persistent the managed object (MO) state change	Down
5	Apply boot stimuli	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/rack-unit-1/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/rack-unit-1/adaptor-1/host-eth-2"/>
```

is composed of the following relative names:

```
topSystem MO: rn="sys"
computeRackUnit MO: rn ="rack-unit-1"
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 initiate and maintain an active session. A successful authentication cookie must be returned by the system before other API calls are allowed.

To authenticate the session, use the following methods:

- aaaLogin—Initializes the session and returns an authentication cookie when successful. The cookie is valid for 600 seconds (10 minutes) and must be refreshed during the session period to prevent it from expiring. A maximum of 4 sessions to CIMC can be opened at any one time.
- aaaRefresh—Creates a new session in place of a previously active session. A new session is created and a new authentication cookie is returned.
- aaaKeepAlive—Maintains the session and keeps the authentication cookie active for another 600 seconds.
- aaaLogout—Exits the current session and deactivates the corresponding authentication cookie.

Operations are performed using the HTTP post method (CIMC supports both HTTP and HTTPS requests) over TCP. HTTP and HTTPS can be configured to use different port numbers, but TCP/80 (or TCP/443 for 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 query methods are supported:

- configResolveDn—Retrieves objects by DN.
- configResolveClass—Retrieves objects of a given class.
- configResolveChildren—Retrieves the child objects of an object.
- configResolveParent—Retrieves the parent object of an object.

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 CIMC 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 CIMC configuration. The resources required for CIMC to process the request can be high, causing CIMC 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 The inRecursive argument and query filters are not supported and are ignored if specified.

Note Until a host is powered on at least once, CIMC may not have complete inventory and status information. For example, if CIMC 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.

Configuration Methods

The CIMC supports only a single method to make configuration changes to managed objects:

- configConfMo—Affects a single managed object (for example, a DN) in the management information tree.

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, CIMC starts sending all new events as they occur.

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 CIMC after 600 seconds of inactivity associated with the event channel session cookie. To prevent automatic closing of the event channel connection by CIMC, 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 CIMC using the same event channel session cookie.

Note The Cisco CIMC XML API sends event notifications only for fault-related events.

Success or Failure Response

When CIMC 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 Requests

When a request has executed successfully, CIMC 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/rack-unit-1/mgmt`:

```
<configResolveDn
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="sys/rack-unit-1/mgmt"/>
```

The response includes the following information:

```
<configResolveDn
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1/mgmt"> <outConfig> <mgmtController
    dn="sys/rack-unit-1/mgmt"
    model="UCS-E160DP-M1/K9"
    serial="FHH16150031"
    subject="blade"
    vendor="Cisco Systems Inc" ></mgmtController></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="sys/rack-unit-1/adaptor-1/ext-eth-0"
     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 CIMC. If the object does not exist, CIMC 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 object:

```
<configResolveDn
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1/adaptor-9999">
```

```
<outConfig>
</outConfig>
</configResolveDn>
```

CHAPTER 2

Using the Cisco CIMC XML API Methods

This chapter includes the following sections:

Authentication Methods

Authentication allows XML API interaction with CIMC. 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 XML API 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 CIMC 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 CIMC 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'>
</aaaLogin>
```

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, CIMC reads the request and performs the actions as provided in the method. CIMC 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="admin">
6 </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 (this can be admin, user, or readonly).

6 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
    cookie="<real_cookie>"
    inCookie="<real_cookie>"
    inName='admin'
    inPassword='password'>
</aaaRefresh>
```

Logging Out of the Session

Use the following method to log out of a session:

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

Unsuccessful Response Examples

Failed login:

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

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

```
<configResolveDn
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1/adaptor-9999">
    <outConfig>
    </outConfig>
</configResolveDn>
```

Bad request:

```
<configConfMo
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1/adaptor-1/ext-eth-0">
    errorCode="103“
    invocationResult="unidentified-fail“
    errorDescr="can't create; object already exists.">
</configConfMo>
```

Query Methods

Using configResolveChildren

When resolving children of objects in the management information tree, 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, on page 21.

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 adapters, use adaptorUnit as the attribute value for classId in the query. This example queries for all instances of the `adaptorUnit` class:

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

See the example request/response in configResolveClass, on page 22.

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, on page 23.

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, on page 25.

CHAPTER 3

Using CIMC XML API Method Descriptions

This chapter includes the following sections:

aaaKeepAlive

The aaaKeepAlive method keeps the session active until the default session time expires, using the same cookie after the method call.

Request Syntax

```
<xs:element name="aaaKeepAlive" type="aaaKeepAlive" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaKeepAlive" mixed="true">
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaKeepAlive" type="aaaKeepAlive" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaKeepAlive" mixed="true">
            <xs:attribute name="cookie" type="xs:string"/>
```

```
                <xs:attribute name="response" type="YesOrNo"/>
                <xs:attribute name="errorCode" type="xs:unsignedInt"/>
                <xs:attribute name="errorDescr" type="xs:string"/>
                <xs:attribute name="invocationResult" type="xs:string"/>
        </xs:complexType>
```

Examples

Request

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

Response

```
<aaaKeepAlive
    cookie="<real_cookie>"
    response="yes">
</aaaKeepAlive>
```

aaaLogin

The aaaLogin method is the login process and is required to begin a session. This action establishes the HTTP (or HTTPS) session between the client and CIMC.

Request Syntax

```
<xs:element name="aaaLogin" type="aaaLogin" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaLogin" mixed="true">
            <xs:attribute name="inName" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:pattern value="[\-\.:_a-zA-Z0-9]{0,16}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inPassword" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:minLength value="0"/>
                         <xs:maxLength value="510"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaLogin" type="aaaLogin" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaLogin" mixed="true">
            <xs:attribute name="outCookie" type="xs:string"/>
            <xs:attribute name="outRefreshPeriod" type="xs:unsignedInt"/>
        <xs:attribute name="outPriv">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:pattern value="(read-only|admin|user){0,1}"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
            <xs:attribute name="outDomains" type="xs:string"/>
            <xs:attribute name="outChannel">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="fullssl"/>
                         <xs:enumeration value="noencssl"/>
                         <xs:enumeration value="plain"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="outEvtChannel">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="fullssl"/>
                         <xs:enumeration value="noencssl"/>
                         <xs:enumeration value="plain"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="outSessionId">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:minLength value="0"/>
                         <xs:maxLength value="32"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="outVersion" type="xs:string"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
        </xs:complexType>
```

Examples

Request

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

Response

```
<aaaLogin
    cookie=""
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin">
</aaaLogin>
```

aaaLogout

The aaaLogout method is a process to close a web session by passing the session cookie as input. It is not automatic; the user has to explicitly invoke the aaaLogout method to terminate the session.

Request Syntax

```
<xs:element name="aaaLogout" type="aaaLogout" substitutionGroup="externalMethod"/>
```

```
        <xs:complexType name="aaaLogout" mixed="true">
            <xs:attribute name="inCookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="cookie" type="stringMin0Max47"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaLogout" type="aaaLogout" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaLogout" mixed="true">
            <xs:attribute name="outStatus">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="success"/>
                         <xs:enumeration value="failure"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
        </xs:complexType>
```

Examples

Request

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

Response

```
<aaaLogout
    cookie="<real_cookie>"
    response="yes"
    outStatus="success">
</aaaLogout>
```

aaaRefresh

The aaaRefresh method keeps sessions active (within the default session time frame) by user activity. There is a default of 600 seconds that counts down when inactivity begins. If the 600 seconds expire, CIMC enters a sleep mode. It requires signing back in, which restarts the countdown. It continues using the same session ID.

Note Using this method expires the previous cookie and issues a new cookie.

Request Syntax

```
<xs:element name="aaaRefresh" type="aaaRefresh" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaRefresh" mixed="true">
            <xs:attribute name="inName" use="required">
```

```
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:pattern value="[\-\.:_a-zA-Z0-9]{0,16}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inPassword" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:minLength value="0"/>
                         <xs:maxLength value="510"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inCookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="cookie" type="stringMin0Max47"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaRefresh" type="aaaRefresh" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaRefresh" mixed="true">
            <xs:attribute name="outCookie" type="xs:string"/>
            <xs:attribute name="outRefreshPeriod" type="xs:unsignedInt"/>
            <xs:attribute name="outPriv">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="(read-only|admin|user){0,1}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="outDomains" type="xs:string"/>
            <xs:attribute name="outChannel">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="fullssl"/>
                         <xs:enumeration value="noencssl"/>
                         <xs:enumeration value="plain"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="outEvtChannel">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="fullssl"/>
                         <xs:enumeration value="noencssl"/>
                         <xs:enumeration value="plain"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
        </xs:complexType>
```

Examples

Request

```
<aaaRefresh
    cookie="<real_cookie>"
    inCookie="<real_cookie>"
    inName='admin'
    inPassword='password'>
```

```
</aaaRefresh>
```

Response

```
<aaaRefresh
    cookie="<real_cookie>"
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin">
</aaaRefresh>
```

configConfMo

The configConfMo method configures the specified managed object in a single subtree (for example, DN).

Request Syntax

```
<xs:element name="configConfMo" type="configConfMo" substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfMo" mixed="true">
            <xs:all>
                <xs:element name="inConfig" type="configConfig" minOccurs="1"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean YesOrNo"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject" use="required"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configConfMo" type="configConfMo" substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfMo" mixed="true">
            <xs:all>
                <xs:element name="outConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Examples

Request

```
<configConfMo
     cookie="<real_cookie>"
     dn="sys/rack-unit-1/sol-if">
     <inConfig><solIf
     dn="sys/rack-unit-1/sol-if"
     adminState="enable"></solIf></inConfig></configConfMo>
```

Response

```
<configConfMo
     dn="sys/rack-unit-1/sol-if"
     cookie="<real_cookie>"
     response="yes">
     <outConfig>
       <solIf dn="sys/rack-unit-1/sol-if"
           adminState="enable"
           name="SoLInterface"
           speed="115200"
           comport="com0"
           status="modified" >
      </solIf>
   </outConfig>
</configConfMo>
```

configResolveChildren

The configResolveChildren method retrieves children of managed objects under a specific DN in the managed information tree. A filter can be used to reduce the number of children being returned.

Request Syntax

```
<xs:element name="configResolveChildren" type="configResolveChildren"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveChildren" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inDn" type="referenceObject" use="required"/>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean YesOrNo"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configResolveChildren" type="configResolveChildren"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveChildren" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </xs:complexType>
```

Examples

Request

```
<configResolveChildren
    cookie="<real_cookie>"
    inDn='sys/rack-unit-1/boot-policy'
    inHierarchical='false'>
</configResolveChildren>
```

Response

```
<configResolveChildren
    cookie="0000227746/06bc6a70-0035-1035-800c-cdac38e14388"
    response="yes"
    dn="sys/rack-unit-1/boot-policy">
    <outConfig>
        <lsbootVirtualMedia
            access="read-write"
            order="2"
            type="virtual-media"
            dn="sys/rack-unit-1/boot-policy/vm-read-write">
        </lsbootVirtualMedia>
        <lsbootLan
            access="read-only"
            order="1" prot="pxe"
            type="lan"
            dn="sys/rack-unit-1/boot-policy/lan-read-only">
        </lsbootLan>
        <lsbootStorage
            access="read-write"
            order="4"
            type="storage"
            dn="sys/rack-unit-1/boot-policy/storage-read-write">
        </lsbootStorage>
        <lsbootEfi
            access="read-only"
            order="3"
            type="efi"
            dn="sys/rack-unit-1/boot-policy/efi-read-only">
        </lsbootEfi>
    </outConfig>
</configResolveChildren>
```

configResolveClass

The configResolveClass method returns requested managed object in a given class. If inHierarchical=true, the results contain children.

Request Syntax

```
<xs:element name="configResolveClass" type="configResolveClass"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveClass" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean YesOrNo"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="classId" type="namingClassId" use="required"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configResolveClass" type="configResolveClass"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveClass" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </xs:complexType>
```

Examples

Request

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

Response

```
<configResolveClass cookie="<real_cookie>"
    response="yes"
    classId="computeRackUnit"> <outConfigs> <computeRackUnit
    dn="sys/rack-unit-1"
    adminPower="policy"
    availableMemory="24576"
    lowVoltageMemory="regular-voltage"
    model="UCS-E160DP-M1/K9"
    memorySpeed="1334"
    name="E160DP"
    numOfAdaptors="0"
    numOfCores="6"
    numOfCoresEnabled="6"
    numOfCpus="1"
    numOfEthHostIfs="0"
    numOfFcHostIfs="0"
    numOfThreads="12"
    operability="operable"
    operPower="off"
    operState="ok"
    originalUuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924"
    presence="equipped"
    serverId="1"
    serial="FHH16150031"
    totalMemory="24576" usrLbl=""
    uuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924"
    vendor="Cisco Systems Inc">
   </computeRackUnit>
  </outConfigs>
</configResolveClass>
```

configResolveDn

The configResolveDn method retrieves a single managed object for a specified DN.

Request Syntax

```
<xs:element name="configResolveDn" type="configResolveDn" substitutionGroup="externalMethod"/>

        <xs:complexType name="configResolveDn" mixed="true">
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean YesOrNo"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject" use="required"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configResolveDn" type="configResolveDn"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveDn" mixed="true">
            <xs:all>
                <xs:element name="outConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Examples

Request

```
<configResolveDn
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="sys/rack-unit-1"/>
```

Response

```
<configResolveDn
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1"> <outConfig>
    <computeRackUnit
        dn="sys/rack-unit-1"
        adminPower="policy"
        availableMemory="24576"
        lowVoltageMemory="regular-voltage"
        model="UCS-E160DP-M1/K9"
        memorySpeed="1334"
        name="E160DP"
        numOfAdaptors="0"
        numOfCores="6"
        numOfCoresEnabled="6"
        numOfCpus="1"
        numOfEthHostIfs="0"
        numOfFcHostIfs="0"
        numOfThreads="12"
        operability="operable"
        operPower="off"
        operState="ok"
        originalUuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924"
        presence="equipped"
        serverId="1"
        serial="FHH16150031"
        totalMemory="24576" usrLbl=""
        uuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924"
        vendor="Cisco Systems Inc">
     </computeRackUnit>
   </outConfig>
</configResolveDn>
```

configResolveParent

For a specified DN, the configResolveParent method retrieves the parent of the managed object.

Request Syntax

```
<xs:element name="configResolveParent" type="configResolveParent"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveParent" mixed="true">
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean YesOrNo"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject" use="required"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configResolveParent" type="configResolveParent"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveParent" mixed="true">
            <xs:all>
                <xs:element name="outConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Examples

Request

```
<configResolveParent
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="sys/rack-unit-1"/>
```

Response

```
<configResolveParent
    cookie="<real_cookie>"
    response="yes"
    dn="sys/rack-unit-1">
    <outConfig>
      <topSystem
        dn="sys"
        address="172.25.209.108"
        currentTime="Mon Jan 7 15:48:57 2013 "
        mode="stand-alone"
        name="ucs-e160dp-m1" >
     </topSystem>
    </outConfig>
</configResolveParent>
```

eventSubscribe

The eventSubscribe method allows a client to subscribe to asynchronous System Event Log (SEL) events generated by CIMC.

Event subscription allows a client application to register for event notification from CIMC. When an event occurs, CIMC informs the client application of the event and its type. Only the actual change information is sent. The object's unaffected attributes are not included.

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

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

Request Syntax

```
<xs:element name="eventSubscribe" type="eventSubscribe" substitutionGroup="externalMethod"/>

        <xs:complexType name="eventSubscribe" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="stringMin0Max47" use="required"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="eventSubscribe" type="eventSubscribe"
substitutionGroup="externalMethod"/>
        <xs:complexType name="eventSubscribe" mixed="true">
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="errorCode" type="xs:unsignedInt"/>
            <xs:attribute name="errorDescr" type="xs:string"/>
            <xs:attribute name="invocationResult" type="xs:string"/>
        </xs:complexType>
```

Examples

Request

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

Response

```
NO RESPONSE OR ACKNOWLEDGMENT.
```

CHAPTER 4

Cisco CIMC XML Schema Definition Files

This chapter includes the following sections:

About the Cisco CIMC XML Schema Definition Files

The Cisco CIMC XML API provides the input XML Schema Definition (xsd) files for every model and a schema definition file for the output:

- RACK-IN.xsd — This document defines the XML document structure for a valid XML request that the CIMC XML API accepts. It also specifies the classes and attributes that you can provide to the XML API configConfMo (Set) requests.
- RACK-OUT.xsd — This document defines the XML document structure for a valid XML response that the CIMC XML API displays. It also specifies the classes and attributes that must appear in the XML API responses.

You can obtain these files from the CIMC at:

RACK-IN.xsd — **https://<CIMC-IP>/visore/RACK-IN.xsd**

RACK-OUT — **https://<CIMC-IP>/visore/RACK-OUT.xsd**

You can use one of the available open source tools to validate the XML document against the XML schema files. In the examples used in this section, xmllint available for download at: www.xmlsoft.org is used as the tool for validation. You also can use the xml schema validation feature of a programming language, for example xerces in Java, for this validation.

Examples of RACK-IN.xsd Usage

Validating an XML Request Using RACK-IN.xsd

Example of Use of the RACK-IN.xsd for an Invalid configResolveClass Request

Request:

```
$cat myXMLRequest.xml
<configResolveClass cookie="1360626069/7189c2b0-d57b-157b-8002-f4759de53d50"
inHierarchical="false"/>
```

Validating the request:

```
/usr/bin/xmllint -schema  ./RACK-IN.xsd  myXMLRequest.xml
<configResolveClass cookie="1360626069/7189c2b0-d57b-157b-8002-f4759de53d50"
inHierarchical="false"/>
```

Response:

```
myXMLRequest.xml:1: element configResolveClass:
Schemas validity error : Element 'configResolveClass': The attribute 'classId' is required
 but missing.
```

In the preceding example, validation of the XML request fails and displays an error because the 'classId' is missing in the request.

Example of Use of the RACK-IN.xsd for a Valid configResolveClass Request

Request:

```
$cat myXMLRequest.xml
<configResolveClass cookie="1360626069/7189c2b0-d57b-157b-8002-f4759de53d50"
inHierarchical="false" classId="topSystem"/>
```

Request:

```
/usr/bin/xmllint -schema ./RACK-IN.xsd  myXMLRequest.xml
<configResolveClass cookie="1360626069/7189c2b0-d57b-157b-8002-f4759de53d50"
inHierarchical="false" classId="topSystem"/>
```

In the preceding example, validation of the XML request for a classResolveClass is successful and the response is displayed.

Example of Use of the RACK-IN.xsd for an Invalid configConfMo Request

Request:

```
$cat setRackUnit.xml
<configConfMo cookie="1300242644/ad04d239-d1aa-498d-b074-ccb923066003"
dn="sys/rack-unit-1" inHierarchical="false">
  <inConfig>
     <computeRackUnit adminPower="down" usrLbl="UCS C240 M3 For Demo"
      availableMemory="16384" dn="sys/rack-unit-1"/>
  </inConfig>
</configConfMo>
```

Validating the request:

```
/usr/bin/xmllint -schema ./RACK-IN.xsd  /setRackUnit.xml
<configConfMo  cookie="1300242644/ad04d239-d1aa-498d-b074-ccb923066003"
dn="sys/rack-unit-1"inHierarchical="false">
  <inConfig>
    <computeRackUnit adminPower="down" usrLbl="UCS C240 M3 For Demo"
     availableMemory="16384" dn="sys/rack-unit-1"/>
  </inConfig>
</configConfMo>
```

Response:

```
/setRackUnit.xml:3: element computeRackUnit: Schemas validity error :
Element 'computeRackUnit', attribute 'availableMemory': The attribute 'availableMemory' is
 not allowed.
/setRackUnit.xml fails to validate
```

The availableMemory attribute is read-only in the computeRackUnit class. You can view the read/write attributes in computeRackUnit that can be set using configCongMo XML by looking at the computeRackUnit definition in RACK-IN.xsd. A sample snippet is as follows:

```
<!--computeRackUnit-->
        <xs:element name="computeRackUnit" type="computeRackUnit"
substitutionGroup="managedObject"/>
        <xs:complexType name="computeRackUnit" mixed="true">

            <xs:attribute name="adminPower">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="up"/>
                         <xs:enumeration value="down"/>
                         <xs:enumeration value="soft-shut-down"/>
                         <xs:enumeration value="cycle-immediate"/>
                         <xs:enumeration value="hard-reset-immediate"/>
                         <xs:enumeration value="bmc-reset-immediate"/>
                         <xs:enumeration value="bmc-reset-default"/>
                         <xs:enumeration value="cmos-reset-immediate"/>
                         <xs:enumeration value="diagnostic-interrupt"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>

            <xs:attribute name="usrLbl">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:pattern value="[
!#$%&amp;\(\)\*\+,\-\./:;\?@\[\]_\{\|\}~a-zA-Z0-9]{0,64}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>

            <xs:attribute name="dn" type="referenceObject"/>
            <xs:attribute name="rn" type="referenceRn"/>
            <xs:attribute name="status" type="objectStatus"/>
        </xs:complexType>
```

Example of Use of the RACK-IN.xsd for an Invalid configConfMo Request

Request:

```
$ cat setBootOrder.xml
<configConfMo cookie="1360205300/79c672f0-d519-1519-8004-30339ee53d50"
inHierarchical="true" dn="sys/rack-unit-1/boot-policy">
  <inConfig>
    <lsbootDef dn="sys/rack-unit-1/boot-policy"  name="boot-policy"  purpose="operational"
 rebootOnUpdate="no">
      <lsbootVirtualMedia access="read-only" order="2" type="virtual-media"
rn="vm-read-only"/>
      <lsbootVirtualMedia access="read-write" order="3" type="virtual-media"
rn="vm-read-write"/>
      <lsbootLan rn="lan-read-only" access="read-only" order="4" prot="pxe" type="lan"/>
      <lsbootStorage rn="storage-read-write" access="read-write" order="1" type="storage">
       <lsbootLocalStorage rn="local-storage"/>
      </lsbootStorage>
      <lsbootEfi rn="efi-read-only" access="read-only" order="5" type="efi"/>
    </lsbootDef>
  </inConfig>
</configConfMo>
```

Validating the request:

```
/usr/bin/xmllint -schema ./RACK-IN.xsd ./setBootOrder.xml
<configConfMo cookie="1360205300/79c672f0-d519-1519-8004-30339ee53d50"
inHierarchical="true" dn="sys/rack-unit-1/boot-policy">
  <inConfig>
    <lsbootDef dn="sys/rack-unit-1/boot-policy" name="boot-policy" purpose="operational"
rebootOnUpdate="no">
      <lsbootVirtualMedia access="read-only" order="2" type="virtual-media"
rn="vm-read-only"/>
      <lsbootVirtualMedia access="read-write" order="3" type="virtual-media"
rn="vm-read-write"/>
      <lsbootLan rn="lan-read-only" access="read-only" order="4" prot="pxe" type="lan"/>
      <lsbootStorage rn="storage-read-write" access="read-write" order="1" type="storage">
        <lsbootLocalStorage rn="local-storage"/>
      </lsbootStorage>
      <lsbootEfi rn="efi-read-only" access="read-only" order="5" type="efi"/>
    </lsbootDef>
  </inConfig>
</configConfMo>
```

Response:

```
./setBootOrder.xml:3: element lsbootDef: Schemas validity error :
Element 'lsbootDef', attribute 'name': The attribute 'name' is not allowed.
./setBootOrder.xml:3: element lsbootDef: Schemas validity error :
Element 'lsbootDef', attribute 'purpose': The attribute 'purpose' is not allowed.
./setBootOrder.xml fails to validate
```

Name and purpose attributes of class lsBootDef are read-only and cannot be used in the configConMo/set request.

Example of Use of the RACK-IN.xsd for a Valid configConfMo Request

Request:

```
usr/bin/xmllint -schema ./RACK-IN.xsd  /setRackUnit.xml
<configConfMo  cookie="1300242644/ad04d239-d1aa-498d-b074-ccb923066003"
 dn="sys/rack-unit-1" inHierarchical="false">
  <inConfig>
    <computeRackUnit adminPower="down" usrLbl="UCS C240 M3 For Demo" dn="sys/rack-unit-1"/>
</inConfig>
</configConfMo>
```

xsd file validates the request and completes the configuration.

Examples of RACK-OUT.xsd Usage

Validating an XML Request Using RACK_OUT.xsd

Example of Use of the RACK-OUT.xsd for a configResolveClass Request

Request:

```
<configResolveClass cookie="1360632361/e892fa10-d57c-157c-8003-f4759de53d50"
inHierarchical="false" classId="computeRackUnit"/>'
https://172.xx.219.xx/nuova | xmllint -format -  |  xmllint -schema ./RACK-OUT.xsd -
```

Response:

```
<configResolveClass cookie="1360632361/e892fa10-d57c-157c-8003-f4759de53d50"
response="yes" classId="computeRackUnit">
  <outConfigs>
    <computeRackUnit dn="sys/rack-unit-1" adminPower="policy" availableMemory="16384"
model="UCSC-C240-M3L"
     memorySpeed="1333" name="UCS C240 M3L" numOfAdaptors="1" numOfCores="16"
numOfCoresEnabled="16" numOfCpus="2"
     numOfEthHostIfs="2" numOfFcHostIfs="3" numOfThreads="32" operPower="on"
     originalUuid="2E5D2295-F32D-48C9-BE8E-BAD36BE174FB" presence="equipped" serverId="1"
serial="FCH1551V030"
    totalMemory="16384" usrLbl="SL2_=+@#$-;,./\" uuid="2E5D2295-F32D-48C9-BE8E-BAD36BE174FB"

     vendor="Cisco Systems Inc"/>
  </outConfigs>
</configResolveClass>
```

Rack-out.xsd validates the output successfully.

Example of Use of the RACK-OUT.xsd for a configResolveClass Request

Request:

```
$ /usr/bin/curl -k -d'<configResolveClass
cookie="1361150931/a5bacff0-d5f5-15f5-8007-f4759de53d50"
inHierarchical="true" classId="topSystem"/>'  https://172.29.219.74/nuova >
C240_complete_MIT.xml
real    0m35.065s
user    0m0.016s
sys     0m0.044s
```

Validating the response:

```
$ ls -l C240_complete_MIT.xml
-rw-r--r--. 1 sajaffer eng58 64905 Feb 17 17:46 C240_complete_MIT.xml

$ /usr/bin/xmllint -schema   RACK-OUT.xsd   C240_complete_MIT.xml 1>/dev/null
C240_complete_MIT.xml validates
```

The preceding XML requests retrieve the complete management information tree of a C-Series server and validate the response against RACK-OUT.xsd

CHAPTER 5

Cisco CIMC XML Object-Access Privileges

This chapter includes the following sections:

Privileges Summary Table

When users are assigned to a role, that role allows certain privileges. Those privileges allow the user access to specific system resources and authorize permission to perform tasks on those resources. The following table lists each privilege and the initial default user role that has been given that privilege.

Internal Name	Label	Description
admin, on page 33	ADMIN	Access to everything
read-only, on page 34	READ_ONLY	Read-only access
user, on page 34	USER	Limited configuration access

Privileges

admin

Purpose

System administration

Responsible Role

Administrator

Controlled Objects

This role is system level. The administrator controls all objects.

read-only

Purpose

Read-only access

Responsible Role

This is not a selectable privilege. All roles have read-only access to all objects. Roles that have read-write privileges on some objects also have read-only access to all other objects.

user

Purpose

Restricted configuration

Responsible Role

User

Controlled Objects

This role can perform the following tasks:

- View all information
- Manage the power control options such as power on, power cycle, and power off
- Launch the KVM console and virtual media
- Clear all logs
- Toggle the locator LED

APPENDIX A

Examples of Common Server Management Tasks

This appendix includes the following topic:

Examples of Common Server Management Tasks

The following examples show how to use the Cisco CIMC XML API to perform common server management tasks. Each example shows the XML API request followed by the response from CIMC.

Retrieving Server Summary Information and Host Power State

Request:

```
<configResolveClass
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    inHierarchical="false"
    classId="computeRackUnit"/>
```

Response:

```
<configResolveClass
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    response="yes"
    classId="computeRackUnit">
   <outConfigs>
      <computeRackUnit dn="sys/rack-unit-1" adminPower="policy" availableMemory="24576"
       lowVoltageMemory="regular-voltage" model="UCS-E160DP-M1/K9"
       memorySpeed="1334" name="E160DP" numOfAdaptors="0" numOfCores="6"
       numOfCoresEnabled="6" numOfCpus="1" numOfEthHostIfs="0" numOfFcHostIfs="0"
       numOfThreads="12" operability="operable" operPower="off" operState="ok"
       originalUuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924" presence="equipped"
       serverId="1" serial="FHH16150031" totalMemory="24576" usrLbl=""
       uuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924" vendor="Cisco Systems Inc">
    </computeRackUnit>
  </outConfigs>
</configResolveClass>
```

Retrieving the Current Running Firmware Versions of Server Components

Request:

```
<configResolveClass
```

```
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    inHierarchical="false"
    classId="firmwareRunning"/>
```

Response:

```
<configResolveClass
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    response="yes"
    classId="firmwareRunning">
  <outConfigs>
   <firmwareRunning
      dn="sys/rack-unit-1/bios/fw-boot-loader" deployment="boot-loader"
      type="blade-bios" version="4.6.4.9" >
  </firmwareRunning>
   <firmwareRunning
      dn="sys/rack-unit-1/mgmt/fw-boot-loader"
      deployment="boot-loader"
      type="blade-controller" version="1.0(1.20130103121559).23">
  </firmwareRunning>
   <firmwareRunning
      dn="sys/rack-unit-1/mgmt/fw-system">
      deployment="system"
      type="blade-controller"> version="1.0(1.20130103121559)" >
  </firmwareRunning>
 </outConfigs> >
</configResolveClass>
```

Retrieving the Backup Firmware Version Installed on CIMC

Request:

```
<configResolveDn
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    inHierarchical="false"
    dn="sys/rack-unit-1/mgmt/fw-updatable"/>
```

Response:

```
<configResolveDn
    cookie="1357577156/9028d030-d2b5-12b5-8007-f289f4c42400"
    response="yes"
    dn="sys/rack-unit-1/mgmt/fw-updatable">
  <outConfig>
   <firmwareUpdatable
      dn="sys/rack-unit-1/mgmt/fw-updatable"
      adminState="triggered"
      deployment="backup"
      operState="ready"
      version="1.0(1.20121206-pha2) #1"
      protocol="none"
      remoteServer=""
      remotePath=""
      user=""
      pwd=""
      progress="0"
      type="blade-controller">
  </firmwareUpdatable>
 </outConfig>
</configResolveDn>
```

Retrieving the Configured Boot Order Table Using the inHierarchical Option

Request:

```
<configResolveClass cookie="1313086522/c7c08988-aa3e-1a3e-8005-5e61c2e14388"
inHierarchical="true" classId="lsbootDef"/>
```

Response:

```
<configResolveClass cookie="1313086522/c7c08988-aa3e-1a3e-8005-5e61c2e14388" response="yes"

      classId="lsbootDef">
  <outConfig>
    <lsbootDef dn="sys/rack-unit-1/boot-policy" name="boot-policy"
          purpose="operational" rebootOnUpdate="no" childAction="deleteNonPresent">
      <lsbootVirtualMedia access="read-only" order="3"
          type="virtual-media" rn="vm-read-only" childAction="deleteNonPresent"/>
      <lsbootVirtualMedia access="read-write" order="5"
          type="virtual-media" rn="vm-read-write" childAction="deleteNonPresent"/>
      <lsbootLan rn="lan-read-only" access="read-only" order="2"
          prot="pxe" type="lan" childAction="deleteNonPresent"/>
      <lsbootStorage rn="storage-read-write" access="read-write" order="1"
          type="storage" childAction="deleteNonPresent">
         <lsbootLocalStorage rn="local-storage" childAction="deleteNonPresent"/>
      </lsbootStorage>
      <lsbootEfi rn="efi-read-only" access="read-only" order="4"
          type="efi" childAction="deleteNonPresent"/>
    </lsbootDef>
  </outConfig>
</configResolveClass>
```

Retrieving the List of Downloaded Host Images

Request:

```
<configResolveClass cookie="1363615849/8e4ceb60-d833-1833-8002-f289f4c42400"
inHierarchical="false" classId="hostImage"/>
```

Response:

```
<configResolveClass cookie="1363615849/8e4ceb60-d833-1833-8002-f289f4c42400" response="yes"

      classId="hostImage">
  <outConfig>
    <hostImage index="1"
         name="linux.iso"
         date="Fri, 15 Mar 2013 04:34:10 GMT"
         size="336222208"
         md5sum="72869d19c2fdea60138d315156b6e7fe"
         dn="sys/rack-unit-1/host-image-mapping/host-image-1"/>
    </hostImage>
  </outConfig>
</configResolveClass>
```

Power Cycling the Server

Request:

```
<configConfMo
    cookie="1357578468/de622490-d2b5-12b5-8009-f289f4c42400"
    dn="sys/rack-unit-1">
  <inConfig>
   <computeRackUnit dn="sys/rack-unit-1"adminPower="cycle-immediate">
  </computeRackUnit>
 </inConfig>
</configConfMo>
```

Response:

```
<configConfMo
    dn="sys/rack-unit-1"
    cookie="1357578468/de622490-d2b5-12b5-8009-f289f4c42400"
    response="yes">
  <outConfig>
 <computeRackUnit
    dn="sys/rack-unit-1" adminPower="policy"
    availableMemory="24576" lowVoltageMemory="NOTINCP-regular-voltage"
    model="UCS-E160DP-M1/K9" memorySpeed="1334" name="E160DP"
    numOfAdaptors="0" numOfCores="6" numOfCoresEnabled="6" numOfCpus="1"
    numOfEthHostIfs="0" numOfFcHostIfs="0" numOfThreads="12"
    operability="NOTINCP-operable" operPower="on" operState="NOTINCP-ok"
    originalUuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924" presence="equipped"
    serverId="1" serial="FHH16150031" totalMemory="24576" usrLbl=""
    uuid="0024C4F4-89F2-0000-A7D1-770BCA4B8924" vendor="Cisco Systems Inc"
    status="modified" >
  </computeRackUnit>
 </outConfig>
</configConfMo>
```

Configuring EFI as the Second Boot Device in the Boot Order Table

Request:

```
<configConfMo cookie="1313090863/ca79ef88-aa3f-1a3f-8006-5e61c2e14388"
     dn="sys/rack-unit-1/boot-policy/efi-read-only" inHierarchical="false">
  <inConfig>
   <lsbootEfi order="2" status="modified" dn="sys/rack-unit-1/boot-policy/efi-read-only"/>

  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/rack-unit-1/boot-policy/efi-read-only"
      cookie="1313090863/ca79ef88-aa3f-1a3f-8006-5e61c2e14388" response="yes">
  <outConfig>
    <lsbootEfi dn="sys/rack-unit-1/boot-policy/efi-read-only" access="read-only" order="2"

      type="efi" status="modified"/>
  </outConfig>
</configConfMo>
```

Removing the Floppy Disk Drive as a Boot Device From the Boot Order List

Request:

```
<configConfMo cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388"
     dn="sys/rack-unit-1/boot-policy/vm-read-write" inHierarchical="true">
  <inConfig>
    <lsbootVirtualMedia order="5" access="read-write" status="deleted"
        dn="sys/rack-unit-1/boot-policy/vm-read-write"/>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/rack-unit-1/boot-policy/vm-read-write"
      cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388" response="yes">
   <outConfig>
   </outConfig>
</configConfMo>
```

Retrieving the SNMP Configuration Details

Request:

```
<configResolveClass cookie="1313086522/c7c08988-aa3e-1a3e-8005-5e61c2e14388"
inHierarchical="false" classId="commSnmp"/>
```

Response:

```
<configResolveClass cookie="1313086522/c7c08988-aa3e-1a3e-8005-5e61c2e14388"
    response="yes" classId="commSnmp">
  <outConfig>
    <commSnmp dn="sys/svc-ext/snmp-svc" adminState="enabled" community="topSecret"
      descr="SNMP Service" name="snmp" port="161" proto="udp" sysContact="demo@demo.com"
      sysLocation="San Jose"/>
  </outConfig>
</configResolveClass>
```

Changing the SNMP Configuration and Retrieving Configured SNMP Trap Receivers Using the inHierarchical Option

Request:

```
<configConfMo cookie="1313090863/ca79ef88-aa3f-1a3f-8006-5e61c2e14388"
      inHierarchical="true" dn="sys/svc-ext/snmp-svc">
 <inConfig>
     <commSnmp dn="sys/svc-ext/snmp-svc" sysContact="TheAdmin@ITDept.com"
        community="demoPrivate" sysLocation="SanJoseCalifornia"/>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/svc-ext/snmp-svc"
     cookie="1313090863/ca79ef88-aa3f-1a3f-8006-5e61c2e14388" response="yes">
  <outConfig>
    <commSnmp dn="sys/svc-ext/snmp-svc" adminState="enabled" community="demoPrivate"
         descr="SNMP Service" name="snmp" port="161" proto="udp"
        sysContact="TheAdmin@ITDept.com" sysLocation="SanJoseCalifornia" status="modified"

         childAction="deleteNonPresent">
      <commSnmpTrap adminState="disabled" community="demoPublic" hostname="11.22.33.44"
         id="1" notificationType="informs" version="v1" rn="snmp-trap-1" status="modified"

         childAction="deleteNonPresent"/>
      <commSnmpTrap adminState="disabled" community="demoPublic" hostname="50.60.70.80"
         id="2" notificationType="informs" version="v1" rn="snmp-trap-2" status="modified"

         childAction="deleteNonPresent"/>
      <commSnmpTrap adminState="disabled" community="demoPublic" hostname="0.0.0.0" id="3"

         notificationType="informs" version="v1" rn="snmp-trap-3" status="modified"
         childAction="deleteNonPresent"/>
      <commSnmpTrap adminState="enabled" community="demoPublic" hostname="138.148.198.218"

         id="4" notificationType="informs" version="v1" rn="snmp-trap-4" status="modified"

         childAction="deleteNonPresent"/>
    </commSnmp>
  </outConfig>
</configConfMo>
```

Retrieving the 'Select Memory RAS' BIOS Token

Request:

```
<configResolveClass cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388"
inHierarchical="false" classId="biosVfSelectMemoryRASConfiguration"/>
```

Response:

```
<configResolveClass cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388"
    response="yes" classId="biosVfSelectMemoryRASConfiguration">
  <outConfig>
    <biosVfSelectMemoryRASConfiguration
       dn="sys/rack-unit-1/bios/bios-settings/SelectMemory-RAS-configuration"
       vpSelectMemoryRASConfiguration="maximum-performance" >
    </biosVfSelectMemoryRASConfiguration>
  </outConfig>
</configResolveClass>
```

Configuring the 'Select Memory RAS' BIOS Token for Mirroring Mode

Request:

```
<configConfMo cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388"
     inHierarchical="false"
     dn="sys/rack-unit-1/bios/bios-settings/SelectMemory-RAS-configuration">
  <inConfig>
    <biosVfSelectMemoryRASConfiguration
       dn="sys/rack-unit-1/bios/bios-settings/SelectMemory-RAS-configuration"
       vpSelectMemoryRASConfiguration="mirroring">
    </biosVfSelectMemoryRASConfiguration>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/rack-unit-1/bios/bios-settings/SelectMemory-RAS-configuration"
     cookie="1313092854/412183f8-aa40-1a40-8007-5e61c2e14388" response="yes">
  <outConfig>
    <biosVfSelectMemoryRASConfiguration
       dn="sys/rack-unit-1/bios/bios-settings/SelectMemory-RAS-configuration"
       vpSelectMemoryRASConfiguration="mirroring" status="modified"/>
  </outConfig>
</configConfMo>
```

Exporting the CIMC Configuration Using TFTP

Request:

```
<configConfMo dn="sys/export-config"
      cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" inHierarchical="false">
  <inConfig>
    <mgmtBackup dn="sys/export-config" adminState="enabled" hostname="198.29.210.14"
       remoteFile="/tftpserver/c250_config_export.cfg"/>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/export-config"
     cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" response="yes">
  <outConfig>
    <mgmtBackup dn="sys/export-config" adminState="disabled"
```

```
        fsmStageDescr="Completed successfully" fsmRmtInvErrCode=""
        fsmRmtInvErrDescr="NONE"
        fsmDescr="export-config" proto="tftp" hostname="" remoteFile=""
        status="modified"/>
  </outConfig>
</configConfMo>
```

The preceding request launches the export operation, which executes as a background task. You can periodically query for the completion status by sending the following request:

Status request:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
inHierarchical="false" classId=" mgmtBackup"/>
```

Status response after completion:

```
<configResolveClass cookie="1313122298/1c207238-aa47-1a47-8009-5e61c2e14388" response="yes"

      classId="mgmtBackup">
  <outConfig>
     <mgmtBackup dn="sys/export-config" adminState="disabled"
       fsmStageDescr="Completed successfully" fsmRmtInvErrCode="" fsmRmtInvErrDescr="NONE"

        fsmDescr="export-config" proto="tftp" hostname="" remoteFile=""/>
  </outConfig>
</configResolveClass>
```

The exported configuration file resembles the following example:

```
[root]# cat /tftpserver/c250_config_export.cfg
<root><cimc>
<version>1.4(0.22)</version>
<network>
<hostname>ucs-c250-M2</hostname>
<mode>dedicated</mode>
<redundancy>active-standby</redundancy>
<dns-use-dhcp>no</dns-use-dhcp>
<preferred-dns-server>0.0.0.0</preferred-dns-server>
<alternate-dns-server>0.0.0.0</alternate-dns-server>
<vlan-enabled>no</vlan-enabled>
.
.
.
```

Importing the CIMC Configuration Using TFTP

Request:

```
<configConfMo dn="sys/import-config"
     cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
     inHierarchical="false">
  <inConfig>
     <mgmtImporter dn="sys/import-config" adminState="enabled"
        hostname="198.29.210.14" remoteFile="/tftpserver/c250_config_export.cfg"/>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/import-config"
     cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" response="yes">
  <outConfig>
      <mgmtImporter dn="sys/import-config" adminState="disabled"
          fsmStageDescr="Error" fsmRmtInvErrCode="" fsmRmtInvErrDescr="NONE"
          fsmDescr="import-config" proto="tftp" hostname="" remoteFile=""
          status="modified"/>
  </outConfig>
</configConfMo>
```

The preceding request launches the import operation, which executes as a background task. You can periodically query for the completion status by sending the following requests:

Status request:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
inHierarchical="false" classId="mgmtImporter"/>
```

Status response before completion:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" response="yes"

     classId="mgmtImporter">
  <outConfig>
    <mgmtImporter dn="sys/import-config" adminState="enabled"
       fsmStageDescr="Applying configuration" fsmRmtInvErrCode=""
       fsmRmtInvErrDescr="NONE" fsmDescr="import-config" proto="tftp" hostname=""
       remoteFile=""/>
  </outConfig>
</configResolveClass>
```

Repeated status request:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
inHierarchical="false" classId="mgmtImporter"/>
```

Status response after completion:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" response="yes"

      classId="mgmtImporter">
  <outConfig>
     <mgmtImporter dn="sys/import-config" adminState="disabled"
        fsmStageDescr="Completed successfully" fsmRmtInvErrCode=""
        fsmRmtInvErrDescr="NONE" fsmDescr="import-config" proto="tftp" hostname=""
        remoteFile=""/>
  </outConfig>
</configResolveClass>
```

Exporting CIMC Technical Support Data Using TFTP

Request:

```
<configConfMo dn="sys/rack-unit-1/tech-support"
      cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" inHierarchical="false">
  <inConfig>
     <sysdebugTechSupportExport dn="sys/rack-unit-1/tech-support" adminState="enabled"
        remoteFile="/tftpserver/c250_techsupport_archive.tgz" hostname="198.29.210.14"/>
  </inConfig>
</configConfMo>
```

Response:

```
<configConfMo dn="sys/rack-unit-1/tech-support"
      cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388" response="yes">
  <outConfig>
     <sysdebugTechSupportExport dn="sys/rack-unit-1/tech-support" adminState="disabled"
         hostname="198.29.210.14" proto="tftp"
         remoteFile="/tftpserver/c250_techsupport_archive.tgz" fsmStageDescr="none"
         fsmProgr="0" fsmStatus="nop" status="modified"/>
  </outConfig>
</configConfMo>
```

The preceding request launches the export operation, which executes as a background task. You can periodically query for the completion status by sending the following requests:

Status request:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
inHierarchical="false" classId="sysdebugTechSupportExport"/>
```

Status response before completion:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
      response="yes" classId="sysdebugTechSupportExport">
  <outConfig>
     <sysdebugTechSupportExport dn="sys/rack-unit-1/tech-support" adminState="enabled"
        hostname="198.29.210.14" proto="tftp"
        remoteFile="/tftpserver/c250_techsupport_archive.tgz"
        fsmStageDescr="collecting" fsmProgr="0" fsmStatus="exporting"/>
  </outConfig>
</configResolveClass>
```

Repeated status request:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
inHierarchical="false" classId="sysdebugTechSupportExport"/>
```

Status response after completion:

```
<configResolveClass cookie="1313118253/2b07f100-aa46-1a46-8008-5e61c2e14388"
      response="yes" classId="sysdebugTechSupportExport">
  <outConfig>
     <sysdebugTechSupportExport dn="sys/rack-unit-1/tech-support" adminState="disabled"
         hostname="198.29.210.14" proto="tftp"
         remoteFile="/tftpserver/c250_techsupport_archive.tgz"
         fsmStageDescr="completed" fsmProgr="100" fsmStatus="success"/>
  </outConfig>
</configResolveClass>
```

The exported technical support file resembles the following example:

```
[root]# tar tvfz /tftpserver/c250_techsupport_archive.tgz | more
drwxr-xr-x root/root         0 2011-08-11 13:01:10 obfl/
-rw-r--r-- root/root     76910 2011-08-11 13:00:56 obfl/obfl-log.1
-rw-r--r-- root/root     76835 1970-01-01 09:38:26 obfl/obfl-log.2
-rw-r--r-- root/root     76881 2011-08-08 21:20:55 obfl/obfl-log.3
-rw-r--r-- root/root     76916 1969-12-31 16:07:28 obfl/obfl-log.4
-rw-r--r-- root/root     76846 2011-08-03 21:38:49 obfl/obfl-log.5
-rw-r--r-- root/root     14598 2011-08-11 20:49:57 obfl/obfl-log
.
.
.
```

APPENDIX B

Notes on Using the configConfMo Method

This appendix includes the following topics:

Defining a Distinguished Name using the configConfMo Method

The **configConfMo** method is used to configure one or more properties in a Managed Object (MO). The MO to be configured is uniquely identified by a Distinguished Name (DN). This chapter shows two ways to provide a DN using the **configConfMo** method.

At the Managed Object Level

You can provide a DN at the Managed Object level. In the following example, the DN "sys/rack-unit-1/sol-if" is defined within the "solIf" MO.

```
<configConfMo
    cookie="<real_cookie>"><inConfig><solIf
    dn="sys/rack-unit-1/sol-if"
    adminState="enable">
   </solIf>
  </inConfig>
</configConfMo>
```

At the Method and Managed Object Level

You can provide a DN at the Method and Managed Object level. In the following example, the DN "sys/rack-unit-1/sol-if" is defined at the **configConfMo** method level and within the "solIf" MO.

```
<configConfMo
    cookie="<real_cookie>"
    dn="sys/rack-unit-1/sol-if">
   <inConfig>
    <solIf
     dn="sys/rack-unit-1/sol-if"
     adminState="enable">
  </solIf>
```

```
 </inConfig>
</configConfMo>
```

Note Specifying a DN at the Method level is optional, and is supported in the Cisco CIMC XML API implementation to be consistent with the Cisco UCS Manager XML API implementation.

Using the Optional inHierarchical Attribute

When a **configConfMo** request is sent to CIMC, the response contains only the immediate properties of the MO being configured.

When the optional inHierarchical attribute is included in the **configConfMo** request, the response will be similar to that of the **configResolveDn** request with the inHierarchical attribute set to true. The response contains the properties for the MO being configured along with the properties of any children MOs.

Request:

```
<configConfMo
    cookie="1357578468/de622490-d2b5-12b5-8009-f289f4c42400"
    inHierarchical="true"
    dn="sys/rack-unit-1/sol-if">
    <inConfig>
      <solIf
         dn="sys/rack-unit-1/sol-if"
         adminState="enable">
      </solIf>
    </inConfig>
</configConfMo>
```

Response:

```
<configConfMo
    dn="sys/rack-unit-1/sol-if"
    cookie="1357578468/de622490-d2b5-12b5-8009-f289f4c42400"
    response="yes">
    <outConfig>
        <solIf
           dn="sys/rack-unit-1/sol-if"
           adminState="enable"
           name="SoLInterface"
           speed="115200"
           comport="com0"
           status="modified"
           childAction="deleteNonPresent">
        </solIf>
   </outConfig>
</configConfMo>
```

Configuring a Single Managed Object

The Cisco implementation accepts only **configConfMo** methods that operate on a single Managed Object (MO). It is invalid to specify a **configConfMo** method that contains multiple MOs even if they are defined in a containment relationship in the CIMC management information model.

The following example shows a valid **configConfMo** method to configure a single MO, "lsbootLan." In this example, the host is configured to use PXE Boot as the first boot option:

```
<configConfMo
```

```
    cookie="<real_cookie>">
    <inConfig>
        <lsbootLan                                             <== Single MO
            order="1"
            status="modified"
            dn="sys/rack-unit-1/boot-policy/lan" >
        </lsbootLan>
    </inConfig>
</configConfMo>
```

The **configConfMo** method in the following example is invalid because a Parent and Child MOs are specified at the same time. The "solIf" MO is a child object of the "computeRackUnit" MO in the management information tree. The Cisco CIMC XML API implementation does not allow a **configConfMo** method to perform subtree configurations.

Request:

```
<configConfMo
    cookie="1313084260/40ea8058-aa3e-1a3e-8004-5e61c2e14388"
    dn="sys/rack-unit-1" inHierarchical="false">
     <inConfig>
         <computeRackUnit                                      <== Parent MO
             adminPower="cycle-immediate"
             usrLbl="Cisco C210 Server"
             dn="sys/rack-unit-1">
                  <solIf                                       <== Child MO
                      dn="sys/rack-unit-1/solif"
                      adminState="enable"
                      speed="9600"/>
         </computeRackUnit>
      </inConfig>
</configConfMo>
```

Response:

```
XML PARSING ERROR: Element 'solif': This element is not expected.
```

Note This method is not supported in the Cisco CIMC XML API implementation but is valid in Cisco UCS Manager XML API implementation.

APPENDIX C

The CIMC Visore Utility

This appendix includes the following topics:

The CIMC Visore Utility

Visore is a utility built into CIMC that allows a user to easily browse Managed Objects (MOs) using an HTML browser. The Visore utility uses the Cisco CIMC XML API query methods to browse the MOs active in CIMC. The Visore utility cannot be used to perform configuration operations.

Accessing Visore

To access Visore, open a browser and enter one of the following URLs:

- http://<CIMC IP Address>/visore.html
- https://<CIMC IP Address>/visore.html

When prompted, log in using the same credentials you would use to log in to the CIMC CLI or GUI user interfaces.

Note In CIMC, only the Firefox and Chrome browsers are supported for Visore access.

Using Visore to Query a Class

To query for a particular class, enter the class name in the **Class or DN** field and click **Run Query**. Visore sends a **configResolveClass** method to CIMC and the requested MO is displayed in a tabular format..

Use the < and > buttons to retrieve the Parent and Child class of the displayed MO. For example, clicking > sends a **configResolveChildren** method to CIMC to query for the child of the MO. Clicking < sends a **configResolveParent** method to CIMC to query for the parent of the MO.

Using Visore to Query a Distinguished Name(DN)

To query for a particular DN, enter the DN in the **Class or DN** field and click **Run Query**. Visore sends a **configResolveDn** method to CIMC.

INDEX