Send document comments to vnmc-docfeedback@cisco.com

Cisco Virtual Network Management Center XML API Reference Guide, Release 1.2

August 9, 2011

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

Text Part Number: OL-25155-01

Cisco Virtual Network Management Center XML API Reference Guide, Release 1.2

CONTENTS

Preface

This preface describes the audience, organization, and conventions of the *Cisco Virtual Network Management Center XML API Reference Guide, Release 1.2*. It also provides information on how to obtain related documentation.

This chapter includes the following sections:

Audience

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

Organization

This guide is organized as follows:

Chapter and Title	Description
Chapter 1, “Introduction to the Cisco VNMC XML API”	Provides an overview of the Cisco VNMC XML API.
Chapter 2, “Using the Cisco VNMC XML API”	Provides a tutorial and details in a use case example.
Chapter 3, “Cisco VNMC XML API Method Descriptions”	Provides descriptions of API methods with syntax and working examples.

Document Conventions

This document uses the following conventions:

Note Means reader *take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Tip Means *the following information will help you solve a problem*.

Command descriptions use these conventions:

Convention	Description
boldface font	Commands and keywords are in boldface.
italic font	Parameters for which you supply values are in italics.
[ ]	Elements in square brackets are optional.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Screen examples use these conventions:

`screen font`	Terminal sessions and information that the switch displays are in screen font.
`boldface screen font`	Information that you must enter is in boldface screen font.
`italic screen font`	Arguments for which you supply values are in italic screen font.
< >	Non-printing characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or number sign (#) at the beginning of a line of code indicates a comment line.

Related Documentation

This section contains information about the documentation available for Cisco Virtual Network Management Center and related products.

This section includes the following topics:

Cisco Virtual Network Management Center Documentation

The following Cisco Virtual Network Management Center documents are available on Cisco.com at the following URL:

http://www.cisco.com/en/US/products/ps11213/tsd_products_support_series_home.html

- *Release Notes for Cisco Virtual Network Management Center, Release 1.2*
- *Cisco Virtual Security Gateway, Release 4.2(1)VSG1(2) and Cisco Virtual Network Management Center, Release 1.2 Installation and Upgrade Guide*
- *Cisco Virtual Network Management Center CLI Configuration Guide, Release 1.2*
- *Cisco Virtual Network Management Center GUI Configuration Guide, Release 1.2*
- *Cisco Virtual Network Management Center XML API Reference Guide, Release 1.2*

Cisco Virtual Security Gateway Documentation

The following Cisco Virtual Security Gateway for the Nexus 1000V Series Switch documents are available on Cisco.com at the following URL:

http://www.cisco.com/en/US/products/ps11208/tsd_products_support_model_home.html

- *Cisco Virtual Security Gateway for Nexus 1000V Series Switch Release Notes, Release 4.2(1)VSG1(2)*
- *Cisco Virtual Security Gateway, Release 4.2(1)VSG1(2) and Cisco Virtual Network Management Center, Release 1.2 Installation and Upgrade Guide*
- *Cisco Virtual Security Gateway for Nexus 1000V Series Switch License Configuration Guide, Release 4.2(1)VSG1(2)*
- *Cisco Virtual Security Gateway for Nexus 1000V Series Switch Configuration Guide, Release 4.2(1)VSG1(2)*
- *Cisco Virtual Security Gateway for Nexus 1000V Series Switch Command Reference, Release 4.2(1)VSG1(2)*
- *Cisco Virtual Security Gateway for Nexus 1000V Series Switch Troubleshooting Guide, Release 4.2(1)VSG1(2)*

Send document comments to vnmc-docfeedback@cisco.com

Cisco Nexus 1000V Series Switch Documentation

The Cisco Nexus 1000V Series switch documentation is available at the following URL:

http://www.cisco.com/en/US/products/ps9902/tsd_products_support_series_home.html

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service. Cisco currently supports RSS Version 2.0.

CHAPTER 1

Introduction to the Cisco VNMC XML API

This chapter provides general information about the Cisco VNMC XML Application Programming Interface (API).

This chapter includes the following sections:

Information About Cisco VNMC and the XML API

This section contains the following topics:

Overview of Cisco VNMC

Cisco Virtual Network Management Center (VNMC) is a virtual appliance that provides centralized device and security policy management for the Cisco Virtual Security Gateway (VSG) and the Cisco Nexus 1000V series switches. Designed for multitenant operation, the Cisco VNMC provides seamless, scalable, and automation-centric management for virtualized data center and cloud environments. With a web-based GUI, CLI, and XML API methods, the Cisco VNMC allows you to manage Cisco VSGs that are deployed throughout the data center from a centralized location.

Multitenancy refers to an architecture principle where multiple client organizations or tenants can have their own virtualized compute, network, and storage resources, which are deployed over a shared physical infrastructure. While multiple tenants may coexist on the same infrastructure, each tenant can have administrative privileges for its virtualized resources. The system is designed to meet the specified SLA for each tenant, including compute, network, storage, and security policies.

The Cisco VNMC is built on an information model-driven architecture, where each managed device is represented by its subcomponents. This architecture enables the Cisco VNMC to provide greater agility and simplification for securing multitenant infrastructure.

VNMC Management Information Model

All the physical and logical components that comprise a Cisco VNMC service component are represented in a hierarchical management information model. This model is referred to as the management information tree (MIT). Each node in the tree represents a managed object (or group of objects) that contains its administrative state and its operational state. The hierarchical structure starts at the top and contains parent and child nodes. Each node in this tree is a managed object and each object has a unique distinguished name (DN) that describes the object and its place in the tree.

Managed objects are abstractions of the Cisco VNMC-managed entities such as policies, rules, security profiles and Cisco VSG instances. Certain managed objects are not created by users but are automatically created by the Cisco VNMC. By invoking the API, objects are read from and written to the MIT. The information model of an individual Cisco VNMC service component is centrally stored and managed by its data management engine (DME). When a user initiates an administrative change to a Cisco VNMC service component (for example, associating a compute firewall profile to a Cisco VSG), the DME first applies that change to the information model, and later the change is applied to the actual Cisco VSG. This approach is called a model-driven framework.

Cisco VNMC Components

The Cisco VNMC consists of multiple service components to provide modularized functionalities for management, tenant management, policy management, resource management, and so on. These components are also called service providers or applications. Each of the components are accessed through a unique URL.

The following sections describe these components:

- Management Controller, page 1-3
- Service Registry, page 1-3
- Resource Manager, page 1-3
- Policy Manager, page 1-4
- VM Manager, page 1-4

Each component has its own data model and a DME (data management engine) to process the model-driven service requests. Each component maintains its own management information tree storage (both in memory and in the persistence store). Data sharing across different service components are achieved with ad-hoc interservice API communications or the publish or subscribe method.

Management Controller

The Management Controller, also known as core service, provides the system-related service for the Cisco VNMC virtual machine. It provides these services:

- Manages user login authentications and authorizations in local or LDAP mode.
- Manages of access controls such as locales, roles and trusted points.
- Administers system information such as the IP address, subnet mask, gateway, and hostname.
- Runs system maintenance such as database backup, data export, and data import.
- Maintains system diagnostic information such as audit logs, faults, event logs, and core dump files.

The Management Controller type is mgmt-controller. Use this service type in the API URL for all Management Controller-related requests.

Service Registry

The Cisco VNMC is designed with a distributed architecture where Service Registry is the central service repository which holds information about all registered managed end-points (VSM, Cisco VSG) and the service providers (Policy Manager, Resource Manager, and so on).

Service endpoints and the service providers register themselves dynamically with the service registry and retrieve the information about interested service components from Service Registry. Service Registry is also responsible for tenant management. It provides these services:

- Creates, deletes, and updates organizations (tenants, data centers, vApps, tiers). Organization changes are automatically propagated to the Policy Manager and Resource Manager where policies and resources are attached to the intended organizations.
- Maintains information about the registered services (providers, endpoints, management controller, service registry)
- Maintains diagnostic information, such as audit logs, faults, and event logs.

The service registry's type is service-reg. Use this service type in the API URL for all Service Registry-related requests.

Resource Manager

The Resource Manager manages logical compute firewalls and their association with actual Cisco VSGs. When a compute firewall is associated with a Cisco VSG, the device configuration profile information (defined by the compute firewall) is pushed to the actual Cisco VSG. This triggers the Cisco VSG to download the security profiles and policies from the Policy Manager. It provides these services:

- Maintains inventory of Cisco Virtual Security Gateways (VSG) and Virtual Supervisor Modules (VSM).
- Defines compute firewalls and associates them with Cisco VSGs for provisioning.
- Integrates with VMWare vCenter instances to retrieve VM attributes.
- Maintains inventory of discovered virtual machines and distributes them to Cisco VSG instances with the following information:
  - VM attributes (name, hypervisor, parent vApp, cluster)
  - vNIC attributes (port profile name, IP addresses)
- Manages pools of Cisco VSGs.

- Maintains health states and faults for VSGs.
- Maintains diagnostic information, such as audit logs, faults, and event logs.

The Resource Manager's type is resource-mgr. Use this service type in the API URL for all Resource Manager-related requests.

Policy Manager

The Policy Manager is the central repository of device configuration profiles, security policies, and all associated artifacts. When a compute firewall is associated with a Cisco VSG from the Resource Manager, the Cisco VSG queries the Policy Manager to resolve the device profile, security profiles, and all referenced policies. The Cisco VSG then configures itself according to the information retrieved from the Policy Manager. Services include the following:

- Defines policies.
- Defines policy sets and assigns policies.
- Defines policy rules with conditions on:
  - Network attributes like protocol, source or destination, or IP address and port
  - Virtual machine attributes like instance name, guest OS, zone, parent app, port profile, cluster, resource pool, hostname, and hypervisor
  - Custom attributes
- Defines zones.
- Defines object groups.
- Defines the security profile dictionary and custom attributes.
- Defines security profiles and assign policy sets.
- Defines firewall device profiles.
- Defines the Cisco VNMC system management device profiles and policies for NTP, DNS, syslog, and faults.
- Distributes policies, security profiles, device profiles, and associated objects to Cisco VSG instances.
- Maintains diagnostic information like audit logs, faults, and event logs.

The Policy Manager's service type is policy-mgr. Use this service type in the API URL for all Policy Manager-related requests.

VM Manager

VM Manager is responsible for interacting with VMWare vCenter and maintaining the VM information retrieved from vCenter. It is a backend service without any user-accessible services. The VM Manager's service type is vm-manager.

Overview of the Cisco VNMC XML API

The Cisco VNMC XML API is a programmatic way to integrate or interact with the Cisco Virtual Network Management Center. The API interface accepts XML documents by using the HTTPS protocol. Developers can use any programming language to generate XML documents that contain the API methods. Configuration and state information is stored in a hierarchical tree structure known as the management information tree (completely exposed through the XML API).

The API model is recursively driven and provides major functionality for application development. For example, changes can be made on a single object, an object subtree, or the entire object tree. Changes can be made to a single attribute on an object or applied to the entire Cisco VNMC structure with a single API call.

The API operates in forgiving mode. Missing attributes are substituted with default values (if applicable) that are maintained in the internal data management engine (DME). If multiple managed objects (that is, policies) are being configured, and any of the managed objects cannot be configured, the API stops its operation, returns the configuration to its prior state, and then stops the API process with a fault notification.

The API leverages an asynchronous operations model to improve scalability and performance. Processes that require time to complete are nonblocking (faster API processes can proceed). A process receives a success message upon a valid request and a complete message when the task is finished.

Full event subscription is supported. The Cisco VNMC sends notifications to all the subscribers for the events (for example, changes to managed objects) that occur, and indicates the type of state change.

Future updates to the managed object data model will conform to the existing object model to ensure backward compatibility. If existing properties are changed during a product upgrade, it will be handled during the database load after the upgrade. New properties will be assigned default values.

The Cisco VNMC uses a model-driven architecture, where changes are first applied to logical constructs in the form of managed objects. The managed objects then apply the changes to the endpoints to achieve the required state (configuration) of the endpoint.

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

The Cisco VNMC API model includes the following programmatic entities:

- Classes—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 (that is, policyDeviceProfile)

A typical request comes into a Cisco VNMC service component's 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. When the request is confirmed, the transactor updates the management information tree. This process is done in a single transaction.

Cisco VNMC Data Model Schema

The data model schema files for all Cisco VNMC service providers are packaged in the Cisco VNMC server and can be retrieved using the following URL:

```
https://<vnmc ip address>/schema
```

The schema list includes:

- core.in.xsd—Management controller configuration APIs and data model
- core.out.xsd—Management controller query APIs and data model
- policy-mgr.in.xsd—Policy manager configuration APIs and data model
- policy-mgr.out.xsd—Policy manager query APIs and data model
- resource-mgr.in.xsd—Resource manager configuration APIs and data model
- resource-mgr.out.xsd—Resource manager query APIs and data model
- service-reg.in.xsd—Service registry configuration APIs and data model
- service-reg.out.xsd—Service registry query APIs and data model

Accessing Cisco VNMC Services

You can access Cisco VNMC services using the XML API request over HTTPS protocol. The HTTPS request URL format is:

```
https://<vnmc ip address>/xmlIM/<service type>
```

<service type> is the service type of the intended service provider as described in the Cisco VNMC Components section. For example, to submit a request to the Policy Manager, use service type policy-mgr in the following URL:

```
https://<vnmc ip address>/xmlIM/policy-mgr
```

Object Naming

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

This section contains the following topics:

- Distinguished Name, page 1-6
- Relative Name, page 1-7

Distinguished Name

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

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

Following is an example:

```
org-root/org-Cisco/zone-trustedServers-0
```

In the example, the DN provides a fully qualified path for a Zone object (zone-trustedServers-0) 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 = "org-root/org-Cisco/zone-trustedServers-0" />
```

Relative Name

The relative name (RN) identifies an object within the context of its parent object. The DN of an object is comprised of the parent DN and the RN in the following format:

```
dn = <parent dn>/{rn}
```

For example, for a Zone object with the name trustedServers-0 defined under the tenant Cisco, its RN is zone-trustedServers-0. Its parent tenant DN is org-root/org-Cisco, and its DN is org-root/org-Cisco/zone-trustedServers-0.

API Method Categories

There are four categories of methods used to interact with the Cisco VNMC. Each API is a method, and each method corresponds to an XML document. The methods are described in these sections:

- Authentication Methods, page 1-7
- Query Methods, page 1-8
- Configuration Methods, page 1-9
- Event Subscription Methods, page 1-10

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 Cisco VNMC 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. Following are the authentication methods:

- aaaLogin—This method is the initial method for logging in to the Cisco VNMC.
- aaaRefresh—This method refreshes the current authentication cookie.
- aaaLogout—This method exits the current session and deactivates the current authentication cookie.

Authentication methods initiate and maintain an active Cisco VNMC session. A successful authentication must be performed before other API calls are allowed. API requests are cookie authenticated.

After a connection session is established and authenticated, a cookie is returned in the response. It is valid for 7200 seconds (120 minutes). The cookie must be refreshed during the session period to prevent it from expiring. Each refresh operation creates a cookie valid for the default interval.

Use aaaLogin to establish a connection session and get a valid cookie. Use aaaRefresh to maintain the session and keep the cookie active. Use aaaLogout to terminate the session (also invalidates the cookie). A maximum of 256 sessions to the Cisco VNMC can be opened at any one time. Subsequent login requests will be rejected after the maximum session limit is reached.

Operations are done using the HTTP post method. The Cisco VNMC supports only the HTTPS protocol on port 443. The HTTP envelope contains the XML configuration.

Query Methods

Query methods obtain information on the current configuration state of a Cisco VNMC object. These 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 objec.t
- 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. Following is an example.

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

API Query methods also might have an inRecursive argument to specify whether the call should be recursive, that is, follow objects that point back to other objects or the parent object.

Query Filters

The API 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. Filters are categorized as:

- Simple filters
- Property filters
- Composite filters
- Modifier filters

Simple Filters

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

- True filter—The resulting set of objects carry the Boolean condition of true.
- False filter—The resulting set of objects carry the Boolean condition of false.

Property Filters

Property filters use the values of an object's properties as the criteria for inclusion in a result set. To create most property filters, the classId and propertyId of the target object and property are 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

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 associationState 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. For example, all faults that occurred between two dates.
- XOR filter—Result set is those objects that pass the filtering criteria of no more than one of the composite's component filters.

Modifier Filters

Modifier filters change the results of a contained filter. Only one modifer filter is supported, the NOT filter. This filter 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 subtrree, or an individual object. Examples of the methods include the following:

- configConfMo—Affects a single subtree, that is, a DN.
- configConfMos—Affects multiple subtrees, that is, several DNs.

- configConfMoGroup—Makes the same configuration changes to multiple subtree structures or managed objects.

Most configuration methods use the argument inHierarchical. 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

When an object is created, changed, or deleted because of a user- or system-initiated action, an event is generated. Applications can get the Cisco VNMC state change information by regular polling or event subscription. Because polling is resource-expensive, event subscription is the preferred method of notification.

Event subscription allows a client application to register for event notification from the Cisco VNMC. When an event occurs, the Cisco VNMC sends the subscribing client applications of the event and its type. Only actual change events are sent, not the object's unaffected attributes. This process applies to all object changes in the system.

To subscribe to the Cisco VNMC event notification, open an HTTP session and keep the session open. Then post the eventSubcribe request through the HTTP session as shown in the following example:

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

After the eventSubscribe request is accepted by the Cisco VNMC, it starts sending all new events as they occur through the HTTP session. To get a valid cookie for event subscription, you must log in to the Cisco VNMC first. If not logged in, the event subscription request is rejected with an error response.

Each event has a unique event ID. These event IDs operate as counters and are included in all event notifications. When an event is generated, the event ID counter increments and a new event is assigned a new event ID. This process enables tracking of events and ensures that no event is missed. If an event is missed by the client, use loggingSyncOcns to retrieve the missed event.

Note For the Cisco VNMC 1.0, only the HTTP protocol is supported for event subscription. Use HTTP when posting the event subcription request.

Capturing XML Interchange Between the Cisco VNMC GUI and the Cisco VNMC Server

The Cisco VNMC GUI is a web-based application. Capture the XML interchange between the GUI and the Cisco VNMC server to learn the real API usage. Because the Cisco VNMC is developed using Adobe FLEX GUI framework, install a debug version of the Adobe flash player. Installation captures the log output stored in a log file under the user's home directory. In Windows 7, the log file can be found under C:\Users\<usename>\AppData\Roaming\Macromedia\Flash Player\Logs\flashlog.txt. For example, most of the sample request and response payload specified in the Cisco VNMC XML API Use Case Example sections are captured in this manner.

Success or Failure Responses

The Cisco VNMC responds almost immediately to any API request. If the request cannot be completed, a failure is returned. For a query or login method, the actual results are returned. For configuration methods, a successful response will indicate that the request is valid, but not indicate that the operation was completed. For example, after a DB backup request is accepted with a successful response from the Cisco VNMC, the Cisco VNMC server might take a longer time to finish the actual backup job.

This section contains the following topics:

Successful Response

Upon a successful response, an XML document is returned with the requested information or a confirmation that changes were made.

This example shows a configResolveDn request on a policy with the DN org-root/org-tenant d3337/pol-pl:

```
<configResolveDn cookie="<real_cookie>"
    dn="org-root/org-tenant_d3337/pol-p1"
    inHierarchical="false"/>
```

The response includes the following details:

```
<configResolveDn
    dn="org-root/org-tenant_d3337/pol-p1"
    cookie="<real_cookie>"
    commCookie="7/13/0/a3c"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfig>
        <policyRuleBasedPolicy
            descr=""
            dn="org-root/org-tenant_d3337/pol-p1"
            intId="10811"
            name="p1"/>
    </outConfig>
</configResolveDn></configResolveDn>
```

Failed Requests

Response to failed requests includes XML attributes for errorCode and errorDescr. This example shows a failed request when trying to create a policy that already exists in the system:

```
<configConfMo
    dn="org-root/org-tenant_d3337/pol-p1"
    cookie="<real_cookie>"
    commCookie="7/13/0/2038"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
```

```
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    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, a success message is returned, but the XML document contains an empty data field, <outConfig> </outConfig>, which indicates that the requested object was not found.

This example shows resolution by DN on a nonexistent policy:

```
<configResolveDn
    dn="org-root/org-tenant_d3337/pol-p1"
    cookie="<real_cookie>"
    commCookie="7/13/0/203e"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfig>
    </outConfig>
</configResolveDn>
```

CHAPTER 2

Using the Cisco VNMC XML API

This chapter includes the following sections:

Methods and Filters

This section contains the following topics:

Authentication Methods

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

Note A session cookie is a 47-character string; it is not the type of cookie that web browsers store locally to maintain session information. Most code examples use <real_cookie> for an actual cookie such as 1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf.

This section contains the following topics:

Login

Example 2-1 uses a Linux POST command to post authentication requests, using HTTPS to connect to Cisco VNMC:

Example 2-1 Login Request

```
POST https://10.193.34.70/xmlIM/mgmt-controller
Please enter content (application/x-www-form-urlencoded) to be POSTed:
<aaaLogin
    inName="admin"
    inPassword="cisco@123"/>
```

Note The XML version and DOCTYPES are not included in aaaLogin and should not be used. 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, the Cisco VNMC reads the request and performs the actions as provided in the method. The Cisco VNMC responds with a message in XML document format and indicates success or failure of the request.

Example 2-2 shows a typical successful response:

Example 2-2 Login Request Response

```
<aaaLogin
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin,read-only"
    outDomains="mgmt02-dummy"
    outChannel="fullssl"
    outEvtChannel="fullssl">
</aaaLogin>
```

Note VNMC 1.0 event channel is using HTTP so it is neither is encrypted or transmitted over SSL

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.

Example 2-3 is the method for refreshing a session:

Example 2-3 Refreshing a Session

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

Logging Out of the Session

Example 2-4 is the method for logging out of a session:

Example 2-4 Logging Out of a Session

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

Response to a Failed Login

Example 2-5 is the response to a failed login:

Example 2-5 Resopnse to a Failed Login

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

Query Methods for Information Gathering

Query methods obtain information that includes hierarchy, state, and scope.

This section contains the following topics:

Using configResolveDn

When resolving a DN, ensure the following:

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

See the example request/response in configResolveDn, page 3-22.

Using configResolveDns

When resolving multiple DNs, ensure the following:

- The objects specified by the DNs are retrieved.
- The specified DN identifies the object instance to be resolved.
- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.
- The order of a request does not determine the order of the response.
- The unknown DNs are returned as part of the outUnresolved attribute.

See the example request/response in configResolveDns, page 3-24.

Using configResolveClass

When resolving a class, ensure the following:

- The objects of the specified class type are retrieved.
- The classId attribute specifies the object class name returned.
- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

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

See the example request/response in configResolveClass, page 3-20.

Using configResolveClasses

When resolving multiple classes, ensure the following:

- The objects of the specified class type are retrieved.
- The classId attribute specifies the object class name returned.
- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

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

See the example request/response in configResolveClasses, page 3-21.

Using configFindDnsByClassId

When finding distinguished names of a specified class, ensure the following

- The DNs of the specified class are retrieved.
- The classId attribute specifies the object type to retrieve.

- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

See the example request/response in configFindDnsByClassId, page 3-16.

Using configResolveChildren

When resolving children of objects in the management information tree, ensure the following:

- The method obtains all child objects of a named object that are instances of the named class.

Note If a class name is omitted, all child objects of the named object are returned.

- The inDn attribute specifies the named object from which the child objects are retrieved.
- The classId attribute specifies the name of the child object class to return.
- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

See the example request/response in configResolveChildren, page 3-18.

Using configResolveParent

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

- The method retrieves the parent object of a specified DN.
- The dn attribute is the DN of the child object.
- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

See the example request/response in configResolveParent, page 3-26.

Using configScope

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

When setting the query scope, ensure the following:

- The method sets the root (scope) of the query to a specified DN and returns objects of the specified class type.
- The dn is the named object from which the query is scoped.
- The inClass attribute specifies the name of the object class to return.

Note When a class is not specified, the query acts the same as configResolveDn.

- The authentication cookie is received from aaaLogin or aaaRefresh.
- The inHierarchical attribute (default = false), if true, specifies that the results are hierarchical.
- The enumerated values, class IDs, and bit masks are displayed as strings.

See the example request/response in configScope, page 3-28.

Query Methods for Policies

Policies are available in different organizations. In a large system with many policies, querying all policies at once is resource intensive. Instead, identify the type of policy and the parent organization to which the policies are attached.

Example 2-6 shows queries for rule-based policies.

Example 2-6 Queries for Rule-Based Policies

```
<configScope
    cookie="<real_cookie>"
    inHierarchical="false"
    dn="org-root/org-tenant_d3338"
    inClass="policyRuleBasedPolicy" />
```

The response is as follows:

```
<configScope
    dn="org-root/org-tenant_d3338"
    cookie="<real_cookie>"
    commCookie="7/13/0/24e7"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <policyRuleBasedPolicy
            descr=""
            dn="org-root/org-tenant_d3338/pol-p9"
            intId="10274"
            name="p9"/>
        <policyRuleBasedPolicy
            descr=""
            dn="org-root/org-tenant_d3338/pol-p1"
            intId="10301"
            name="p1"/>
    </outConfigs>
</configScope>
```

The following query using the DN is more efficient:

```
<configResolveDn
    inHierarchical="false"
    cookie="<real_cookie>"
    dn="sys org-root/org-tenant_d3338/pol-p1">
</configResolveDn>
```

To get the policy object and its rule data, change inHierarchical to true:

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

```
    dn=" org-root/org-tenant_d3338/pol-p1">
</configResolveDn>
```

Query Methods for Faults

Example 2-7 shows queries for faults.

Example 2-7 Queries for Faults

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

This example, which contains the filter <inFilter> eq class= “faultInst”, obtains a list of all major or critical faults:

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

Filters

Use filters to create specific and targeted queries.

This section contains the following topics:

Simple Filters

Simple filters use true and false conditions.

This section contains the following topics:

False Conditions

This example shows a simple filter using false conditions:

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

```
    inHierarchical="false">
    <inFilter>
    </inFilter>
</configResolveClass>
```

True Conditions

This example shows a simple filter using true conditions:

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

Property Filters

This section contains the following topics:

Equality Filter

This example shows an equality filter:

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

Not Equal Filter

This example shows a not equal filter:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="fwComputeFirewall">
    <inFilter>
        <ne class="fwComputeFirewall"
```

```
            property="assocState"
            value="associated" />
    </inFilter>
</configResolveClass>
```

Greater Than Filter

This example shows a greater than filter:

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

Greater Than or Equal to Filter

This example shows a greater than or equal to filter:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="sysdebugCore">
    <inFilter>
        <ge class="sysdebugCore"
            property="size"
            value="2097152" />
    </inFilter>
</configResolveClass>
```

Less Than Filter

This example shows a less than filter:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="sysdebugCore">
    <inFilter>
        <lt class="sysdebugCore"
            property="size"
            value="2097152" />
    </inFilter>
</configResolveClass>
```

Less Than or Equal to Filter

This example shows a less than or equal to filter::

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="sysdebugCore">
    <inFilter>
        <le class="sysdebugCore"
            property="size"
            value="2097152" />
    </inFilter>
```

```
</configResolveClass>
```

Wildcard Filter

This example shows a wildcard filter that finds any virtual firewall whose name begins with the prefix “dmz”:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="fwInstance">
    <inFilter>
        <wcard class="fwInstance"
            property="name"
            value="dmz.*" />
    </inFilter>
</configResolveClass>
```

Any Bits Filter

This example shows an any bits filter:

```
<configResolveClass
    cookie="null"
    inHierarchical="false"
    classId="extpolClient">
    <inFilter>
        <anybit class="extpolClient"
            property="capability"
            value="vm-fw,vm-vasw" />
    </inFilter>
</configResolveClass>
```

All Bits Filter

This example shows an all bits filter:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="extpolClient">
    <inFilter>
        <allbits class="extpolClient"
            property="capability"
            value="vm-fw,vm-vasw" />
    </inFilter>
</configResolveClass>
```

Composite Filters

This section contains the following topics:

AND Filter

This example shows an and filter that finds firewalls that are associated with a configuration:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="fwComputeFirewall">
    <inFilter>
        <and>
            <eq class="fwComputeFirewall"
                property="assocState"
                value="associated" />
            <eq class="fwComputeFirewall"
                property="configState"
                value="applied" />
        </and>
    </inFilter>
</configResolveClass>
```

OR Filter

This example shows an or filter that returns all managed endpoints whose operational state is either lost-visibility or unregistered.

```
<configResolveClass
    inHierarchical="false"
    cookie="<real_cookie>"
    classId="extpolClient">
    <inFilter>
        <or>
            <eq class="extpolClient"
                property="operState"
                value="unregistered"/>
            <eq class="extpolClient"
                property="operState"
                value="lost-visibility"/>
        </or>
    </inFilter>
</configResolveClass>
```

Between Filter

This example shows a between filter that finds memory arrays with slots 1, 2, 3, 4, or 5 populated:

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

AND OR NOT Composite Filters

This example shows an AND OR NOT composite filter that returns all objects of type computeBlade that are located in slots 1 or 8 in all chassis, except for chassis 5:

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

Modifier Filters

NOT Filter

This example shows a not modifier filter:

```
<configResolveClass
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="extpolClient">
    <inFilter>
        <not>
            <anybit class="extpolClient"
                property="capability"
                value="vm-fw" />
        </not>
    </inFilter>
</configResolveClass>
```

API Examples

This section provides API configuration and management examples.

This section contains the following topics:

Authentication Using the Management Controller

Access to Cisco VNMC is session-based and must first be authenticated with a login request. Default session lengths are 120 minutes. Sessions can be extended with the aaaKeepAlive and aaaRefresh methods. We recommend that when activity is completed that the user manually log out of the session using aaaLogout. Use service type mgmt-controller to send login requests to the Cisco VNMC.

This section contains the following topics:

Authentication Request

This example shows an authentication request:

```
POST URL:  https://10.193.33.221/xmlIM/mgmt-controller
XML API payload:
<aaaLogin
    inName="admin"
    inPassword="Nbv12345"/>
```

Authentication Response

This examples shows an authentication response:

```
<aaaLogin
    cookie=""
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc="" destSvc=""
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin,read-only"
    outDomains=""
    outChannel="fullssl"
    outEvtChannel="fullssl"
    outSessionId="web_13528"
    outVersion="1.0">
</aaaLogin>
```

Tenant Management Using the Service Registry

The Service Registry, service type service-reg, creates and manages tenants and the suborganizations that are used to organize the policies and resources. These tenants and suborganizations are arranged in a tree hierarchy for the bottom-up policy and resource resolution. Class names used in tenant and suborganization creation support the following four levels:

- Tenant, class orgTenant—Defines the top level organization.
- Data Center, class orgDatatcenter—Defines a data center under a Tenant.
- Application, class orgApp—Defines an application under a Data Center.
- Tier, class orgTier—Defines a tier under an Application.

To create a new organization, or update an existing organization, provide the following:

- Object DN
- Attribute values
- Status equal to created or modified

To delete an organization, use a status = deleted in the request.

This section contains the following topics:

- Create or Update Organization Request, page 2-14
- Create or Update Organization Response, page 2-14

Create or Update Organization Request

This example shows how to create a tenant named demoTenant.

```
POST URL:  https://10.193.33.221/xmlIM/service-reg
<configConfMos
    cookie="<real_cookie>">
  <inConfigs>
    <pair key="org-root/org-demoTenant">
        <orgTenant dn="org-root/org-demoTenant"
                               name="demoTenant"
                               status="created"/>
    </pair>
  </inConfigs>
</configConfMos>
```

Create or Update Organization Response

This example shows the response that is received after creating a tenant named demoTenant.

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="2/15/0/839"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="service-reg_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-demoTenant">
            <orgTenant
                descr=""
                dn="org-root/org-demoTenant"
                fltAggr="0"
                level="1"
                name="demoTenant"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Policy Management

Note In each example where the configConfMos API uses a single XML element <pair> to specify a single configuration object, the configConfMo API can be used as an alternative to obtain the same result.

This section includes the following topics:

Device Policies

This section includes the following topics:

Syslog Policy

A syslog policy object tracks all actions that take place within the system. It sets the logging level for a device policy and creates mysyslog. Example 2-8 shows a syslog policy request.

Example 2-8 Syslog Policy Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/syslog-mysyslog">
            <commSyslog dn="org-root/syslog-mysyslog">
            <commSyslogConsole
                adminState="enabled"
                severity="emergencies"/>
            <commSyslogClient
                name="primary"
                adminState="enabled"
                hostname="5.6.7.8"
                severity="notifications"
                forwardingFacility="local7"/>
            <commSyslogClient
                name="secondary"
                adminState="enabled"
                hostname="123.23.53.123"
                severity="warnings"
                forwardingFacility="local5"/>
            </commSyslog>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1b9"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
        <outConfigs>
            <pair key="org-root/syslog-mysyslog">
            <commSyslog
                adminState="enabled"
                descr=""
                dn="org-root/syslog-mysyslog"
                intId="25301"
                name="mysyslog"
                port="514"
                proto="udp"
                severity="warnings"
                status="created"/>
            </pair>
    </outConfigs>
</configConfMos>
```

SNMP Policy

This example creates an SNMP policy named **mysnmp**. Once created, this policy is available by that name in the device profile listing.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/snmp-mysnmp">
        <commSnmp dn="org-root/snmp-mysnmp"
            adminState="enabled"
            descr="The default SNMP policy"
            sysContact="Andrew Jackson"
            sysLocation="San Jose" >
        <commSnmpCommunity
            community="public"
            role="read-only"/>
        <commSnmpTrap
            hostname="nms-1.cisco.com"
            community="private"/>
        <commSnmpTrap
            hostname="nms-2.cisco.com"
            community="private"/>
        </commSnmp>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1bc"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
```

```
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/snmp-mysnmp">
        <commSnmp
            adminState="enabled"
            descr="The default SNMP policy"
            dn="org-root/snmp-mysnmp"
            intId="25281"
            name="mysnmp"
            port="161"
            proto="all"
            sysContact="Andrew Jackson"
            sysLocation="San Jose"/>
        </pair>
    </outConfigs>
</configConfMos>
```

LogProfile Policy

This example sets the debug level and logging level for a LogProfile policy named **debugLog**.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/logprof-default">
        <policyLogProfile
            dn="org-root/logprof-debugLog"
            name="debugLog"
            level="debug1"
            size="10000000"
            backupCount="4"/>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/33"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/logprof-debugLog">
            <policyLogProfile
            adminState="enabled"
            backupCount="4"
            descr="the log level for every process"
            dn="org-root/logprof-debugLog"
            intId="10068"
            level="debug1"
            name="debugLog"
            size="10000000"
            status="modified"/>
        </pair>
    </outConfigs>
```

```
</configConfMos>
```

Device Profiles

This action creates a device profile named **myDeviceProfile**. It enables the profile and designates the contents of the profile.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-Cola/fwdevprofile-myDeviceProfile">
        <fwpolicyFirewallDeviceProfile
            dn="org-root/org-Cola/fwdevprofile-myDeviceProfile"
            adminState="enabled"
            snmpPolicy="mysnmp"
            syslogPolicy="mysyslog"
            timezone="America/Los_Angeles" >
        <commDns
            name="somedns.com"
            domain="somedns.com"/>
<!-- The order means the device should first use the DNS provider with the smallest
"order" value. If that DNS server is not responsive, use the DNS provider with the next
smaller number. -->
        <commDnsProvider
            hostip="171.70.168.183"
            order="1"/>
        <commDnsProvider
            hostip="171.68.226.120"
            order="2"/>
        <commDnsProvider
            hostip="64.102.6.247"
            order="3"/>
        <commNtpProvider
            name="somentp.com"
            order="1"/>
        <commNtpProvider
            name="north-america.pool.ntp.org"
            order="2"/>
        </fwpolicyFirewallDeviceProfile>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1ba"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-Cola/fwdevprofile-myDeviceProfile">
        <fwpolicyFirewallDeviceProfile
            adminState="enabled"
            coreFilePolicy=""
            descr=""
```

```
            dn="org-root/org-Cola/fwdevprofile-myDeviceProfile"
            dnsPolicy=""
            enablePolicyDecisionLog="no"
            faultPolicy=""
            httpPolicy=""
            httpsPolicy=""
            intId="25326"
            logProfilePolicy=""
            name="myDeviceProfile"
            snmpPolicy="mysnmp"
            status="created"
            syslogPolicy="mysyslog"
            telnetPolicy=""
            timezone="America/Los_Angeles"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Zone

This example creates two zones named **trustedClients-0** and **trustedServers-0**. A set of attributes with values is also assigned.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-Cisco/zone-trustedClients-0">
<!-- Create a zone of all VMs in the 192.168.1.0/24 subnet -->
            <policyZone dn="org-root/org-Cisco/zone-trustedClients-0">
                <policyZoneCondition id="1" order="1">
                    <policyZoneExpression opr="prefix">
                        <policyIPAddress id="1" value="192.168.1.0" />
                            <policyIPSubnet id="1" value="255.255.255.0" />
                    </policyZoneExpression>
                </policyZoneCondition>
            </policyZone>
        </pair>
        <pair key="org-root/org-Cisco/zone-trustedServers-0">
<!-- Create a zone of all VMs attached to a VNSP where the "appType" is set to
"BuildServer" -->
            <policyZone dn="org-root/org-Cisco/zone-trustedServers-0">
                <policyZoneCondition id="1" order="1">
                    <policyZoneExpression opr="eq">
                        <policyParentAppName id="1" value="BuildServer" />
                    </policyZoneExpression>
                </policyZoneCondition>
            </policyZone>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1a6"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
```

```
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-tenant0/zone-zone0">
            <policyZone
                descr=""
                dn="org-root/org-tenant0/zone-zone0"
                intId="24295"
                name="zone0"
                status="created"/>
        </pair>
        <pair key="org-root/org-Cisco/zone-trustedServers-0">
            <policyZone
                descr=""
                dn="org-root/org-Cisco/zone-trustedServers-0"
                intId="24404"
                name="trustedServers-0"
                status="created"/>
        </pair>
        <pair key="org-root/org-Cisco/zone-trustedClients-0">
            <policyZone
                descr=""
                dn="org-root/org-Cisco/zone-trustedClients-0"
                intId="24408"
                name="trustedClients-0"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Object Group

This example creates an object group named **ftpPorts0** and assigns a set of attributes.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-tenant0/objgrp-ftpPorts0">
            <policyObjectGroup dn="org-root/org-tenant0/objgrp-ftpPorts0">
                <policyObjectGroupExpression id="1" order="1" opr="eq">
                    <policyNetworkPort id="1" value="20" />
                </policyObjectGroupExpression>
                <policyObjectGroupExpression id="2" order="2" opr="eq">
                    <policyNetworkPort id="1" value="21" />
                </policyObjectGroupExpression>
            </policyObjectGroup>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1a4"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
```

```
    response="yes">
    <outConfigs>
        <pair key="org-root/org-tenant0/objgrp-ftpPorts0">
            <policyObjectGroup
                attributeName=""
                descr=""
                dn="org-root/org-tenant0/objgrp-ftpPorts0"
                intId="24265"
                name="ftpPorts0"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Attribute Dictionary

This example sets the dictionary that defines the various attribute IDs and names.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
<!-- Create Sample VnspCustomDictionary under org-Cisco -->
        <pair key="org-root/attr-dict-custom-userAttrs">
            <policyVnspCustomDictionary dn="org-root/attr-dict-custom-userAttrs">
                <policyVnspCustomAttr dataType="string" id="1" name="userAttr1" />
                <policyVnspCustomAttr dataType="string" id="2" name="userAttr2" />
                <policyVnspCustomAttr dataType="string" id="3" name="dept" />
                <policyVnspCustomAttr dataType="string" id="4" name="production" />
            </policyVnspCustomDictionary>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1a3"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/attr-dict-custom-userAttrs">
            <policyVnspCustomDictionary
                descr=""
                dn="org-root/attr-dict-custom-userAttrs"
                intId="24245"
                name="userAttrs"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Policy

This example creates a policy named **trustedHosts** and sets the rules it can use.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-Cisco/pol-trustedHosts">
            <policyRuleBasedPolicy dn="org-root/org-Cisco/pol-trustedHosts">
                <policyRule name="allowSsh" order="1">
<!-- This rule allows all VMs in zone "trustedClients" to initiate an SSH connection to
VMs in zone "trustedServers" -->
                <policyRuleCondition id="100" order="1">
                    <policyNetworkExpression opr="eq">
                    <policyNwAttrQualifier attrEp="source"/>
                    <policyZoneNameRef id="1" value="trustedClients-0" />
                    </policyNetworkExpression>
                </policyRuleCondition>
                <policyRuleCondition id="101" order="20">
                    <policyNetworkExpression opr="eq">
                    <policyNwAttrQualifier attrEp="destination"/>
                    <policyZoneNameRef id="1" value="trustedServers-0" />
                    </policyNetworkExpression>
                </policyRuleCondition>
                <policyRuleCondition id="103" order="30">
                    <policyNetworkExpression opr="eq">
                    <policyNwAttrQualifier attrEp="destination"/>
                    <policyNetworkPort id="1" placement="0" value="22" />
                    </policyNetworkExpression>
                </policyRuleCondition>
                <fwpolicyAction actionType="permit"/>
                </policyRule>
                <policyRule name="allowTacacs" order="2">
                <fwpolicyAction actionType="permit"/>
                </policyRule>
            </policyRuleBasedPolicy>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1b5"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-Cisco/pol-trustedHosts">
            <policyRuleBasedPolicy
                descr=""
                dn="org-root/org-Cisco/pol-trustedHosts"
                intId="25131"
                name="trustedHosts"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

PolicySet

This example creates **myPolicySet0** and sets the order that policies are applied.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-tenant0/pset-myPolicySet0">
            <policyPolicySet
                dn="org-root/org-tenant0/pset-myPolicySet0"
<!-- Policy Set contains references to three policies. Policies are resolved by name in
the org hierarchy. Ordering of policies is important, so we use the "order" property to
order the policies. Note that two policy sets can refer to the same  policies, and use a
different order. -->
            <policyPolicyNameRef order="1" policyName="trustedHosts"/>
            </policyPolicySet>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookied>"
    commCookie="7/15/0/1a8"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-tenant0/pset-myPolicySet0">
            <policyPolicySet
                adminState="enabled"
                descr=""
                dn="org-root/org-tenant0/pset-myPolicySet0"
                intId="24431"
                name="myPolicySet0"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Security Profile

This example creates a security profile named **vnsp-sp1** and assigns the VNSP to it.

Request

```
POST URL:  https://10.193.33.221/xmlIM/policy-mgr
XML API payload:
<configConfMos
cookie="<real_cookie>">
<inConfigs>
<pair key="org-root/org-tenant0/vnsp-sp1">
    <policyVirtualNetworkServiceProfile
        dn="org-root/org-tenant0/vnsp-sp1"
        policySetNameRef="myPolicySet0">
        <policyVnspAVPair id="1">
```

```
            <policyAttributeDesignator attrName="dept"/>
            <policyAttributeValue value="DEV" />
        </policyVnspAVPair>
        </policyVirtualNetworkServiceProfile>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1ac"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/org-tenant0/vnsp-sp1">
            <policyVirtualNetworkServiceProfile
                descr=""
                dn="org-root/org-tenant0/vnsp-sp1"
                intId="24512"
                name="sp1"
                policySetNameRef="myPolicySet0"
                status="created"
                vnspId="2"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Resource Management

The Resource Management component performs the following functions:

- Assigns the compute firewall to a device policy.
- Queries for available firewall instances.
- Associates firewall instances with a managed object.

This section includes the following topics:

Compute Firewall

This example assigns **computeFirewall** to a device policy.

Request

```
POST URL:  https://10.193.33.221/xmlIM/resource-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-Cisco/cfw-VSNaaa">
```

```
            <fwComputeFirewall dn="org-root/org-Cisco/cfw-VSNaaa"
                devicePolicy="myDeviceProfile"
                hostname="cfw-VSNaaa"
                dataIpAddress="10.10.10.100">
            </fwComputeFirewall>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="5/15/0/19a"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
     <outConfigs>
        <pair key="org-root/org-Cisco/cfw-VSNaaa">
            <fwComputeFirewall
                assocState="unassociated"
                configState="not-applied"
                dataIpAddress="10.10.10.100"
                dataIpSubnet="255.255.255.0"
                descr=""
                devicePolicy="myDeviceProfile"
                dn="org-root/org-Cisco/cfw-VSNaaa"
                fltAggr="0"
                hostname="cfw-VSNaaa"
                intId="12368"
                name="VSNaaa"
                status="created"/>
        </pair>
    </outConfigs>
</configConfMos>
```

Query Firewall Instances

This example queries all firewall instances and their attributes.

Request

```
POST URL:  https://10.193.33.221/xmlIM/resource-mgr
XML API payload:
<configResolveClass
    cookie="<real_cookie>"
    classId="fwInstance"
    inHierarchical="false">
    <inFilter>
        <eq class="fwInstance"
            property="mgmtIp"
            value="10.193.33.221" />
    </inFilter>
</configResolveClass>
```

Response

```
configResolveClass
    cookie=<real_cookie>"
    commCookie="5/15/0/1d9"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
```

```
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes"
    classId="fwInstance">
    <outConfigs>
        <fwInstance
            assignedToDn=""
            assoc="none"
            descr=""
            dn="fw/inst-1005"
            fltAggr="0"
            fsmDescr=""
            fsmPrev="DisassociateSuccess"
            fsmProgr="100"
            fsmRmtInvErrCode="none"
            fsmRmtInvErrDescr=""
            fsmRmtInvRslt=""
            fsmStageDescr=""
            fsmStamp="2010-12-03T23:18:13.304"
            fsmStatus="nop"
            fsmTry="0"
            intId="10155"
            mgmtIp="10.193.33.221"
            model=""
            name="firewall"
            pooled="0"
            registeredClientDn="extpol/reg/clients/client-1005"
            revision="0"
            serial=""
            svcId="1005"
            vendor=""/>
    </outConfigs>
</configResolveClass>
```

Associate Firewall Instance

This example associates a firewall instance to a managed object and assigns firewall-0 to the inst-1005.

Request

```
POST URL:  https://10.193.33.221/xmlIM/resource-mgr
XML API payload:
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/org-Cola/cfw-firewall-0">
            <fwComputeFirewall dn="org-root/org-Cola/cfw-firewall-0">
            <fwResourceBinding assignedToDn="fw/inst-1005"/>
            </fwComputeFirewall>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="5/15/0/1df"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
```

```
    <outConfigs>
        <pair key="org-root/org-Cola/cfw-firewall-0">
            <fwComputeFirewall
                assocState="associated"
                configState="not-applied"
                dataIpAddress="168.1.1.221"
                dataIpSubnet="255.255.255.0"
                descr=""
                devicePolicy="myDeviceProfile"
                dn="org-root/org-Cola/cfw-firewall-0"
                fltAggr="1"
                hostname="cfw-firewall-0"
                intId="12514"
                name="firewall-0"
                status="modified"/>
        </pair>
    </outConfigs>
</configConfMos>
```

CHAPTER 3

Cisco VNMC XML API Method Descriptions

This chapter includes the following:

API method details are provided below.

Cisco VNMC XML API Methods

These methods are also called from the GUI console. This section provides API method descriptions, syntax (request and response), and a usage example. The API methods for the Cisco VNMC are defined as follows:

- aaaGetRemoteUserRoles
- aaaGetUserLocales
- aaaKeepAlive
- aaaLogin
- aaaLogout
- aaaRefresh
- configConfFiltered
- configConfMo
- configConfMoGroup
- configConfMos
- configFindDnsByClassId
- configConfMoGroup
- configMoChangeEvent
- configResolveClass
- configResolveClasses
- configResolveDn
- configResolveDns
- configResolveParent
- configScope
- eventSendHeartbeat
- eventSubscribe
- eventSubscribeApps
- faultAckFault
- faultAckFaults
- faultResolveFault
- loggingSyncOcns
- methodVessel
- orgResolveElements
- orgResolveInScope
- orgResolveLogicalParents
- policyEstimateImpact
- poolResolveInScope

aaaGetRemoteUserRoles

This example returns user privileges for the remote location.

Request Syntax

```
<xs:element name="aaaGetRemoteUserRoles" type="aaaGetRemoteUserRoles"
substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaGetRemoteUserRoles" mixed="true">
            <xs:attribute name="inRemoteUserName">
```

```
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                             <xs:pattern value="[a-zA-Z][a-zA-Z0-9_.@-]{0,31}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="cookie" type="xs:string"/>
                <xs:attribute name="response" type="YesOrNo"/>
            </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaGetRemoteUserRoles" type="aaaGetRemoteUserRoles"
substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaGetRemoteUserRoles" mixed="true">
            <xs:attribute name="outRemoteUserPriv">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:pattern
value="((policy|aaa|read-only|admin|tenant|operations|res-config|fault),){0,7}(policy|aaa|
read-only|admin|tenant|operations|res-config|fault){0,1}"/>
                </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

```
<aaaGetRemoteUserRoles
    cookie="<real_cookie>"
    inRemoteUserName="adminuser"
    outRemoteUserPriv/>
```

Response

```
<aaaGetRemoteUserRoles
    cookie="<real_cookie>"
    commCookie="11/15/0/2964"
    srcExtSys="10.193.33.109"
    destExtSys="10.193.33.109"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes"
    outRemoteUserPriv="admin">
</aaaGetRemoteUserRoles>
```

aaaGetUserLocales

This example returns a list of authorized user locales.

Request Syntax

```
<xs:element name="aaaGetUserLocales" type="aaaGetUserLocales"
substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaGetUserLocales" mixed="true">
            <xs:attribute name="inUserName">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[a-zA-Z][a-zA-Z0-9_.@-]{0,31}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inIsUserRemote">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="aaaGetUserLocales" type="aaaGetUserLocales"
substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaGetUserLocales" mixed="true">
            <xs:attribute name="outUserLocales">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="0"/>
                        <xs:maxLength value="512"/>
                    </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

```
<aaaGetUserLocales
    cookie="<real_cookie>"
    inUserName="john"
    inIsUserRemote="no"
    outUserLocales/>
```

Response

```
<aaaGetUserLocales
```

```
    cookie="<real_cookie>"
    commCookie="11/15/0/2962"
    srcExtSys="10.193.33.109"
    destExtSys="10.193.33.109"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes"
    outUserLocales="TestSanity">
</aaaGetUserLocales>
```

aaaKeepAlive

This example keeps the session active until the default session time expires and uses 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="xs:string"/>
            <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>" />
```

Response

```
<aaaKeepAlive
    cookie="<real_cookie>"
    commCookie="11/15/0/2969"
    srcExtSys="10.193.33.109"
    destExtSys="10.193.33.109"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes">
</aaaKeepAlive>
```

aaaLogin

This example shows the login process that is required to begin a session and which establishes an authenticated HTTPS session between the client and the VNMC.

Request Syntax

```
<xs:element name="aaaLogin" type="aaaLogin" substitutionGroup="externalMethod"/>
        <xs:complexType name="aaaLogin" mixed="true">
            <xs:attribute name="inName">
                <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">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                         <xs:minLength value="0"/>
                         <xs:maxLength value="512"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <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="((policy|aaa|read-only|admin|tenant|operations|res-config|fault),){0,7}(policy|aaa|
read-only|admin|tenant|operations|res-config|fault){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="1"/>
                     <xs:maxLength value="64"/>
                </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="Nbv12345"/>
```

Response

```
<aaaLogin cookie=""
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc="" destSvc=""
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin"
    outDomains=""
    outChannel="fullssl"
    outEvtChannel="fullssl"
    outSessionId="web_49019"
    outVersion="1.0(0.39938)">
</aaaLogin>
```

aaaLogout

This example shows the logout process to end a current session. When the default session time period expires, the process is called automatically.

Request Syntax

```
<xs:element name="aaaLogout" type="aaaLogout" substitutionGroup="externalMethod"/>
    <xs:complexType name="aaaLogout" mixed="true">
        <xs:attribute name="inCookie" type="xs:string"/>
        <xs:attribute name="cookie" type="xs:string"/>
        <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
    inCookie="<real_cookie>"
    outStatus/>
```

Response

```
<aaaLogout cookie=""
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc=""
    destSvc=""
    response="yes"
    outStatus="success">
</aaaLogout>
```

aaaRefresh

Sessions can be kept active (within the default session time frame) by user activity. There is a default of 7200 seconds that counts down when inactivity begins. If the 7200 seconds expire, the Cisco VNMC enters a sleep mode and requires signing back in, which restarts the countdown. The session 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">
                <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">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="0"/>
                        <xs:maxLength value="512"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inCookie" type="xs:string"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <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="((policy|aaa|read-only|admin|tenant|operations|res-config|fault),){0,7}(policy|aaa|
read-only|admin|tenant|operations|res-config|fault){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>"
    inName="admin"
    inPassword="Nbv12345"
    inCookie="<real_cookie>"/>
```

Response

```
<aaaRefresh
    cookie="<real_cookie>"
    commCookie="" srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc=""
    destSvc=""
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="admin"
    outDomains=""
    outChannel="fullssl"
    outEvtChannel="fullssl">
</aaaRefresh>
```

configConfFiltered

This example shows how data and activity are limited according to the configured policies.

Request Syntax

```
<xs:element name="configConfFiltered" type="configConfFiltered"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfFiltered" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
                <xs:element name="inConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </xs:complexType>
```

Response Syntax

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

```
<configConfFiltered
    cookie="<real_cookie>"
    inHierarchical="false"
    classId="orgTenant">
    <inFilter>
        <eq class="orgTenant"
            property="name"
            value="Cisco" />
    </inFilter>
    <inConfig>
        <orgDatacenter
            dn="org-HR"
            descr="HR (Human Resources- new Descr)"/>
    </inConfig>
</configConfFiltered>
```

Response

```
<configConfFiltered
    cookie="<real_cookie>"
    commCookie="5/15/0/617"
    srcExtSys="10.193.33.206"
    destExtSys="10.193.33.206"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes"
    classId="orgTenant">
    <outConfigs>
        <orgDatacenter
            descr="HR (Human Resources- new Descr)"
            dn="org-root/org-Cisco/org-HR"
            fltAggr="0"
            level="2"
            name="HR"
            status="modified"/>
    </outConfigs>
</configConfFiltered>
```

configConfMo

This example shows how the specified Managed Object is configured 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="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </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
    dn=""
    cookie="<real_cookie>"
    inHierarchical="false">
    <inConfig>
        <aaaLdapEp
            attribute="CiscoAvPair"
            basedn="dc=pasadena,dc=cisco,dc=com"
            descr=""
            dn="sys/ldap-ext"
            filter="sAMAccountName=$userid"
            retries="1"
            status="modified"
            timeout="30"/>
    </inConfig>
</configConfMo>
```

Response

```
<configConfMo
    dn=""
    cookie="<real_cookie>"
    commCookie="11/15/0/28"
    srcExtSys="10.193.33.101"
    destExtSys="10.193.33.101"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes">
    <outConfig>
        <aaaLdapEp
            attribute="CiscoAvPair"
            basedn="dc=pasadena,dc=cisco,dc=com"
            childAction="deleteNonPresent"
            descr=""
            dn="sys/ldap-ext"
            filter="sAMAccountName=$userid"
            fsmDescr=""
            fsmPrev="updateEpSuccess"
            fsmProgr="100"
            fsmStageDescr=""
            fsmStamp="2010-11-22T23:41:01.826"
            fsmStatus="nop"
            fsmTry="0"
            intId="10027"
            name=""
            retries="1"
            status="modified"
            timeout="30"/>
    </outConfig>
</configConfMo>
```

configConfMoGroup

This example shows how groups of managed objects are configured based upon the configured policies.

Request Syntax

```
<xs:element name="configConfMoGroup" type="configConfMoGroup"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfMoGroup" mixed="true">
            <xs:all>
                <xs:element name="inDns" type="dnSet" minOccurs="0"/>
                <xs:element name="inConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
```

```
        </xs:complexType>
```

Response Syntax

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

Examples

Note The descr property of orgDataCenter (under org-root/org-Cisco and org-root/org-Cola) is modified. Because the descr property is not implicit, it can be modified. If implicit, the modification does not apply and a new orgDataCenter is created.

Request

```
<configConfMoGroup
    cookie="<real_cookie>"
    inHierarchical="false">
    <inDns>
        <dn value="org-root/org-Cisco" />
        <dn value="org-root/org-Cola" />
    </inDns>
    <inConfig>
        <orgDatacenter
            dn="org-HR"
            descr="HR (Human Resources)"/>
    </inConfig>
</configConfMoGroup>
```

Response

```
<configConfMoGroup
    cookie="<real_cookie>"
    commCookie="5/15/0/600"
    srcExtSys="10.193.33.206"
    destExtSys="10.193.33.206"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
    <outConfigs>
        <orgDatacenter
            descr="HR (Human Resources)"
            dn="org-root/org-Cola/org-HR"
            fltAggr="0"
            level="2"
            name="HR"
            status="modified"/>
        <orgDatacenter
            descr="HR (Human Resources)"
```

```
            dn="org-root/org-Cisco/org-HR"
            fltAggr="0"
            level="2"
            name="HR"
            status="modified"/>
    </outConfigs>
</configConfMoGroup>
```

configConfMos

This example shows how to configure managed objects in multiple subtrees using DNs.

Request Syntax

```
<xs:element name="configConfMos" type="configConfMos" substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfMos" mixed="true">
            <xs:all>
                <xs:element name="inConfigs" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_2">
                        <xs:selector xpath="pair"/>
                        <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configConfMos" type="configConfMos" substitutionGroup="externalMethod"/>
        <xs:complexType name="configConfMos" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_4">
                        <xs:selector xpath="pair"/>
                        <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </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:complexType>
```

Examples

Request

```
<configConfMos
    cookie="<real_cookie>">
    <inConfigs>
        <pair key="org-root/logprof-default">
            <policyLogProfile dn="org-root/logprof-default"
                name="default"
                level="debug1"
                size="10000000"
                backupCount="4"/>
        </pair>

    <!-- Update Controller Device Profile -->
        <pair key="org-root/controller-profile-default">
            <policyControllerDeviceProfile
                dn="org-root/controller-profile-default"
                adminState="enabled">

                <commDnsProvider hostip="171.70.168.183" order="1"/>
                <commDnsProvider hostip="171.68.226.120" order="2"/>
                <commDnsProvider hostip="64.102.6.247"   order="3"/>
            </policyControllerDeviceProfile>
        </pair>
    </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
    cookie="<real_cookie>"
    commCookie="7/15/0/1a74"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="org-root/logprof-default">
            <policyLogProfile
                adminState="enabled"
                backupCount="4"
                descr="the log level for every process"
                dn="org-root/logprof-default"
                intId="10065"
                level="debug1"
                name="default"
                size="10000000"/>
        </pair>
        <pair key="org-root/controller-profile-default">
            <policyControllerDeviceProfile
                adminState="enabled"
                coreFilePolicy=""
                descr="default profile for management server virtual machine"
                dn="org-root/controller-profile-default"
                dnsPolicy=""
                faultPolicy="default"
                httpPolicy="default"
                httpsPolicy="default"
                intId="10057"
                logProfilePolicy="default"
                name="default"
```

```
                snmpPolicy=""
                syslogPolicy=""
                telnetPolicy=""
                timezone=""/>
        </pair>
    </outConfigs>
</configConfMos>
```

configFindDnsByClassId

This example shows how to find distinguished names and return them sorted by class ID.

Request Syntax

```
<xs:element name="configFindDnsByClassId" type="configFindDnsByClassId"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configFindDnsByClassId" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configFindDnsByClassId" type="configFindDnsByClassId"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configFindDnsByClassId" mixed="true">
            <xs:all>
                <xs:element name="outDns" type="dnSet" 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

```
<configFindDnsByClassId
    classId="computeBlade"
    cookie="<real_cookie>" />
```

Response

```
<configFindDnsByClassId
    cookie="<real_cookie>"
    response="yes"
    classId="computeBlade">
    <outDns>
        <dn value="sys/chassis-1/blade-7"/>
        <dn value="sys/chassis-1/blade-5"/>
```

```
        <dn value="sys/chassis-1/blade-3"/>
        <dn value="sys/chassis-1/blade-1"/>
    </outDns>
</configFindDnsByClassId>
```

configMoChangeEvent

This example shows how to return change events on the specified managed object.

Request Syntax

```
<xs:element name="configMoChangeEvent" type="configMoChangeEvent"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configMoChangeEvent" mixed="true">
            <xs:all>
                <xs:element name="inConfig" type="configConfig" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inEid" type="xs:unsignedLong"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

```
<configMoChangeEvent
    cookie=""
    commCookie=""
    srcExtSys="10.193.33.120"
    destExtSys="10.193.33.120"
    srcSvc="service-reg_dme"
    destSvc="sam_extXMLApi"
    inEid="71449">
    <inConfig>
        <extpolRegistry
            dn="extpol/reg"
            lastPollTs="2010-11-12T20:33:51.071"
            status="modified"/>
    </inConfig>
</configMoChangeEvent>
```

Response

```
Event (12:35:20:675):
<eventSendHeartbeat
```

```
    cookie="0/0/0/2a74"
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc="" destSvc=""
    response="yes"
    outSystemTime="2010-11-12T20:34:19.630">
</eventSendHeartbeat>

Event (12:35:20:690):
<eventSendHeartbeat
    cookie="0/0/0/2a74"
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc="" destSvc=""
    response="yes"
    outSystemTime="2010-11-12T20:34:19.630">
</eventSendHeartbeat>
```

configResolveChildren

This example shows how to retrieve 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"/>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <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>"
    classId="aaaUser"
    inDn="sys/user-ext"
    inHierarchical="false">
    <inFilter>
    </inFilter>
</configResolveChildren>
```

Response

```
<configResolveChildren
    cookie="<real_cookie>"
    commCookie="11/15/0/2a59"
    srcExtSys="10.193.33.120"
    destExtSys="10.193.33.120"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes"
    classId="aaaUser">
    <outConfigs>
        <aaaUser descr="" dn="sys/user-ext/user-doe"
            email="" expiration="never" expires="no" firstName="John" intId="12999"
            lastName="Doe" name="doe" phone="" priv="admin,read-only" pwdSet="yes"/>
        <aaaUser descr="" dn="sys/user-ext/user-jacks" email="" expiration="never"
            expires="no" firstName="Play" intId="12734" lastName="Jacks" name="jacks"
            phone="" priv="fault,operations,policy,read-only,res-config,tenant"
            pwdSet="yes"/>
        <aaaUser descr="" dn="sys/user-ext/user-admin" email="" expiration="never"
            expires="no" firstName="" intId="10052" lastName="" name="admin" phone=""
            priv="admin,read-only" pwdSet="yes"/>
        <aaaUser descr="" dn="sys/user-ext/user-over" email="" expiration="never"
            expires="no" firstName="Roll" intId="12711" lastName="Over" name="over"
            phone="" priv="fault,operations,policy,read-only,res-config,tenant"
            pwdSet="yes"/>
        <aaaUser descr="" dn="sys/user-ext/user-fun" email="" expiration="never"
            expires="no" firstName="Have" intId="12708" lastName="Fun" name="fun" phone=""
            priv="read-only" pwdSet="yes"/>
        <aaaUser descr="testuser" dn="sys/user-ext/user-aaa" email="" expiration="never"
            expires="no" firstName="a" intId="10620" lastName="aa" name="aaa" phone=""
            priv="aaa,read-only" pwdSet="no"/>
    </outConfigs>
</configResolveChildren>
```

configResolveClass

This example shows how to return the requested managed object in a given class. If *inHierarchical* = true, the object contains 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">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="classId" type="namingClassId"/>
        </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>"
    classId="pkiEp"
    inHierarchical="false">
    <inFilter>
    </inFilter>
</configResolveClass>
```

Response

```
<configResolveClass
    cookie="<real_cookie>"
    commCookie="11/15/0/2a5b"
    srcExtSys="10.193.33.120"
    destExtSys="10.193.33.120"
    srcSvc="sam_extXMLApi"
    destSvc="mgmt-controller_dme"
    response="yes"
    classId="pkiEp">
    <outConfigs>
        <pkiEp descr=""
            dn="sys/pki-ext"
            intId="10037"
            name=""/>
    </outConfigs>
</configResolveClass>
```

configResolveClasses

This example shows how to return requested managed objects in several classes. If *inHierarchical* = true, the object contains children.

Request Syntax

```
<xs:element name="configResolveClasses" type="configResolveClasses"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveClasses" mixed="true">
            <xs:all>
                <xs:element name="inIds" type="classIdSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

Examples

Request

```
<configResolveClasses
    cookie="<real_cookie>"
    inHierarchical="false">
    <inIds>
        <Id value="computeBlade"/>
        <Id value="equipmentChassis"/>
    </inIds>
</configResolveClasses>
```

Response

(This is an abbreviated response.)

```
<configResolveClasses
    cookie="<real_cookie>"
    response="yes">
    <outConfigs>
        <computeBlade
            adminPower="policy"
            adminState="in-service"
            dn="sys/chassis-1/blade-1"
            .
            .
        <computeBlade
            adminPower="policy"
            adminState="in-service"
            dn="sys/chassis-1/blade-3"
            .
            .
        <computeBlade
            adminPower="policy"
            adminState="in-service"
            dn="sys/chassis-1/blade-5"
            .
            .
        <computeBlade
            adminPower="policy"
            adminState="in-service"
            dn="sys/chassis-1/blade-7".
            .
            .
        <equipmentChassis
            adminState="acknowledged"
            configState="ok"
            .
            .
    </outConfigs>
</configResolveClasses>
```

configResolveDn

This example shows how to retrieve 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">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </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>"
    dn="vmmEp/vm-mgr-vcenter1" />
```

Response

```
<configResolveDn dn="vmmEp/vm-mgr-vcenter1"
    cookie="<real_cookie>"
    commCookie="9/15/0/1c0d"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="vm-mgr_dme"
    response="yes">
    <outConfig>
        <vmManager
            adminState="enable"
            descr=""
            dn="vmmEp/vm-mgr-vcenter1"
```

```
            fltAggr="0"
            fsmDescr="AG registration with
vCenter(FSM:sam:dme:VmManagerRegisterWithVCenter)"
            fsmPrev="RegisterWithVCenterRegistering"
            fsmProgr="13"
            fsmRmtInvErrCode="none"
            fsmRmtInvErrDescr=""
            fsmRmtInvRslt=""
            fsmStageDescr="AG registration with
vCenter(FSM-STAGE:sam:dme:VmManagerRegisterWithVCenter:Registering)"
            fsmStamp="2010-11-11T21:37:15.696"
            fsmStatus="RegisterWithVCenterRegistering"
            fsmTry="1"
            hostName="savbu-vpod-dev-31.cisco.com"
            intId="21959"
            name="vcenter1"
            operState="unknown"
            stateQual=""
            type="vmware"
            version=""/>
    </outConfig>
</configResolveDn>
```

configResolveDns

This example shows how to retrieve managed objects for a list of DNs.

Request Syntax

```
<xs:element name="configResolveDns" type="configResolveDns"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveDns" mixed="true">
            <xs:all>
                <xs:element name="inDns" type="dnSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configResolveDns" type="configResolveDns"
substitutionGroup="externalMethod"/>
        <xs:complexType name="configResolveDns" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configSet" minOccurs="0"/>
                <xs:element name="outUnresolved" type="dnSet" 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:complexType>
```

Examples

Request

```
<configResolveDns
    cookie="<real_cookie>"
    inHierarchical="false">
    <inDns>
        <dn value="sys/chassis-1" />
        <dn value="sys/chassis-1/blade-1/board/cpu-1" />
        <dn value="sys/chassis-1/blade-1/board/t-stats" />
    </inDns>
</configResolveDns>
```

Response

```
<configResolveDns
    cookie="<real_cookie>"
    response="yes">
    <outConfigs>
        <processorUnit
            arch="Xeon"
            cores="4"
            dn="sys/chassis-1/blade-1/board/cpu-1"
            id="1"
            model="Intel(R) Xeon(R) CPU E5520 @ 2.27GHz"
            operState="operable"
            operability="operable"
            perf="not-supported"
            power="not-supported"
            presence="equipped"
            revision="0"
            serial=""
            socketDesignation="CPU1"
            speed="2.266000"
            stepping="5"
            thermal="ok"
            threads="8"
            vendor="Intel(R) Corporation"
            voltage="ok"/>
        <equipmentChassis
            adminState="acknowledged"
            configState="ok"
            connPath="A,B"
            connStatus="A,B"
            dn="sys/chassis-1"
            fabricEpDn="fabric/server/chassis-1"
            fltAggr="2"
            fsmDescr=""
            fsmPrev="PsuPolicyConfigSuccess"
            fsmProgr="100"
            fsmRmtInvErrCode="none"
            fsmRmtInvErrDescr=""
            fsmRmtInvRslt=""
            fsmStageDescr=""
```

```
                fsmStamp="2009-09-13T21:34:37"
                fsmStatus="nop"
                fsmTry="0"
                id="1"
                lcTs="1969-12-31T16:00:00"
                managingInst="A"
                model="N20-C6508"
                operQualifier="fabric-conn-problem"
                operState="fabric-conn-problem"
                operability="operable"
                power="ok"
                presence="unknown"
                revision="0"
                serial="FOX1252GG84"
                thermal="ok"
                vendor="Cisco Systems Inc"
                versionHolder="yes"/>
    </outConfigs>
    <outUnresolved>
        <dn value="sys/chassis-1/blade-1/board/t-stats"/>
    </outUnresolved>
</configResolveDns>
```

configResolveParent

This example shows how to retrieve the parent of the managed object for a specified DN.

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">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </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/chassis-1/blade-1/adaptor-1">
</configResolveParent>
```

Response

```
<configResolveParent dn="sys/chassis-1/blade-1/adaptor-1"
    cookie="<real_cookie>"
    response="yes">
    <outConfig>
        <computeBlade
            adminPower="policy"
            adminState="in-service"
            assignedToDn=""
            association="none"
            availability="available"
            availableMemory="10240"
            chassisId="1"
            checkPoint="discovered"
            connPath="A,B"
            connStatus="A,B"
            descr=""
            diagnostics="complete"
            discovery="complete"
            dn="sys/chassis-1/blade-1"
            fltAggr="0"
            fsmDescr=""
            fsmFlags=""
            fsmPrev="DiscoverSuccess"
            fsmProgr="100"
            fsmRmtInvErrCode="none"
            fsmRmtInvErrDescr=""
            fsmRmtInvRslt=""
            fsmStageDescr=""
            fsmStamp="2009-09-23T23:44:30"
            fsmStatus="nop"
            fsmTry="0"
            intId="768052"
            lc="discovered"
            lcTs="1969-12-31T16:00:00"
            managingInst="B"
            model="N20-B6620-1"
            name=""
            numOfAdaptors="1"
            numOfCores="8"
            numOfCpus="2"
            numOfEthHostIfs="2"
            numOfFcHostIfs="0"
            numOfThreads="16"
            operPower="off"
            operQualifier=""
```

```
                operState="unassociated"
                operability="operable"
                originalUuid="e3516842-d0a4-11dd-baad-000bab01bfd6"
                presence="equipped"
                revision="0"
                serial="QCI12520024"
                slotId="1"
                totalMemory="10240"
                uuid="e3516842-d0a4-11dd-baad-000bab01bfd6"
                vendor="Cisco Systems Inc"/>
        </outConfig>
</configResolveParent>
```

configScope

This example shows how to return managed objects and details about their configuration.

Request Syntax

```
<xs:element name="configScope" type="configScope" substitutionGroup="externalMethod"/>
        <xs:complexType name="configScope" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inClass" type="namingClassId"/>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inRecursive">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>

            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="configScope" type="configScope" substitutionGroup="externalMethod"/>
        <xs:complexType name="configScope" 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="dn" type="referenceObject"/>
        </xs:complexType>
```

Examples

Request

```
<configScope
    dn="org-root"
    cookie="<real_cookie>"
    inClass="orgOrgRes"
    inHierarchical="false"
    inRecursive="false">
    <inFilter>
    </inFilter>
</configScope>
```

Response

```
<configScope dn="org-root"
    cookie="<real_cookie>"
    commCookie="2/15/0/2a53"
    srcExtSys="10.193.33.120"
    destExtSys="10.193.33.120"
    srcSvc="sam_extXMLApi"
    destSvc="service-reg_dme"
    response="yes">
    <outConfigs>
        <orgOrgCaps
            dn="org-root/org-caps"
            org="512"
            tenant="64"/>
        <orgOrgCounts
            dn="org-root/org-counter"
            org="36"
            tenant="7"/>
    </outConfigs>
</configScope>
```

eventSendHeartbeat

This example shows how to send an event that indicates the current session is still active.

Request Syntax

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

Response Syntax

```
<xs:element name="eventSendHeartbeat" type="eventSendHeartbeat"
substitutionGroup="externalMethod"/>
        <xs:complexType name="eventSendHeartbeat" mixed="true">
            <xs:attribute name="outSystemTime" type="dateTime"/>
            <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

When the client application subscribes to an event or events using eventSubscribeApps or eventSubscribe, the Cisco VNMC sensd eventSendHeartbeat periodically (default 120 seconds).

Response

```
<eventSendHeartbeat cookie="0/0/0/2a76"
    commCookie=""
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc=""
    destSvc=""
    response="yes"
    outSystemTime="2010-11-12T20:38:19.630">
</eventSendHeartbeat>
```

eventSubscribe

This example shows how to send a subscribe request for activity.

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="xs:string"/>
            <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>">
    <inFilter>
    </inFilter>
</eventSubscribe>
```

Response

```
NO RESPONSE OR ACKNOWELDGEMENT.
```

eventSubscribeApps

This example shows a subscribe request for activity on specified applications. The client application can subscribe to the Cisco VNMC system to receive events from different applications. In eventApplication, ip is the IP address for the VM where the application (DME) is running. The client application can subscribe to receive events from the Cisco VSG as well, where *ip* should be the IP address for the Cisco VSG, and type is managed-endpoint.

Request Syntax

```
<xs:element name="eventSubscribeApps" type="eventSubscribeApps"
substitutionGroup="externalMethod"/>
        <xs:complexType name="eventSubscribeApps" mixed="true">
            <xs:all>
                <xs:element name="inAppList" type="configSet" minOccurs="0"/>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

```
<eventSubscribeApps
    cookie="<real_cookie>"
    commCookie=""
```

```
    srcExtSys="0.0.0.0"
    destExtSys="0.0.0.0"
    srcSvc=""
    destSvc="">
    <inAppList>
        <eventApplication
            ip="10.193.33.101"
            type="service-reg"/>
            <eventApplication
            ip="10.193.33.101"
            type="policy-mgr"/>
        <eventApplication
            ip="10.193.33.101"
            type="mgmt-controller"/>
    </inAppList>
    <inFilter>
    </inFilter>
</eventSubscribeApps>
```

Response

```
IF SUCCESSFUL, NO RESPONSE OR ACKNOWELDGEMENT.
```

faultAckFault

This example shows how to send an acknowledgement when a fault is recorded.

Request Syntax

```
<xs:element name="faultAckFault" type="faultAckFault" substitutionGroup="externalMethod"/>
        <xs:complexType name="faultAckFault" mixed="true">
            <xs:attribute name="inId" type="xs:unsignedLong"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

```
<faultAckFault
    inHierarchical="false"
    cookie="<real_cookie>"
    inId="10120" />
```

Response

```
<faultAckFault
    cookie="<real_cookie>"
    commCookie="5/15/0/6c"
    srcExtSys="10.193.33.214"
    destExtSys="10.193.33.214"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
</faultAckFault>
```

faultAckFaults

This example shows how to send an acknoweldgement when multiple faults are recorded.

Request Syntax

```
<xs:element name="faultAckFaults" type="faultAckFaults"
substitutionGroup="externalMethod"/>
        <xs:complexType name="faultAckFaults" mixed="true">
            <xs:all>
                <xs:element name="inIds" type="idSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

```
<faultAckFaults
    cookie="<real_cookie>">
    <inIds>
        <id value="10656"/>
        <id value="10660"/>
    </inIds>
</faultAckFaults>
```

Response

```
<faultAckFaults
    cookie="<real_cookie>"
    commCookie="11/15/0/505"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
```

```
    destSvc="mgmt-controller_dme"
    response="yes">
</faultAckFaults>
```

faultResolveFault

This example shows how to send a response when a fault has been resolved.

Request Syntax

```
<xs:element name="faultResolveFault" type="faultResolveFault"
substitutionGroup="externalMethod"/>
        <xs:complexType name="faultResolveFault" mixed="true">
            <xs:attribute name="inId" type="xs:unsignedLong"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="faultResolveFault" type="faultResolveFault"
substitutionGroup="externalMethod"/>
        <xs:complexType name="faultResolveFault" mixed="true">
            <xs:all>
                <xs:element name="outFault" 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:complexType>
```

Examples

Request

```
<faultResolveFault
    inHierarchical="false"
    cookie="<real_cookie>"
    inId="10120" />
```

Response

```
<faultResolveFault
    cookie="<real_cookie>"
    commCookie="5/15/0/6a"
    srcExtSys="10.193.33.214"
    destExtSys="10.193.33.214"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
    <outFault>
        <faultInst
            ack="yes"
            cause="empty-pool"
            changeSet=""
            code="F0135"
```

```
                created="2010-11-19T11:02:41.568"
                descr="Virtual Security Gateway pool default is empty"
                dn="org-root/fwpool-default/fault-F0135"
                highestSeverity="minor"
                id="10120"
                lastTransition="2010-11-19T11:02:41.568"
                lc=""
                occur="1"
                origSeverity="minor"
                prevSeverity="minor"
                rule="fw-pool-empty"
                severity="minor"
                tags=""
                type="equipment"/>
    </outFault>
</faultResolveFault>
```

loggingSyncOcns

This example shows how to retrieve event IDs from DME.

Request Syntax

```
<xs:element name="loggingSyncOcns" type="loggingSyncOcns"
substitutionGroup="externalMethod"/>
        <xs:complexType name="loggingSyncOcns" mixed="true">
            <xs:attribute name="inFromOrZero" type="xs:unsignedLong"/>
            <xs:attribute name="inToOrZero" type="xs:unsignedLong"/>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="loggingSyncOcns" type="loggingSyncOcns"
substitutionGroup="externalMethod"/>
        <xs:complexType name="loggingSyncOcns" mixed="true">
            <xs:all>
                <xs:element name="outStimuli" type="MethodSet" 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:complexType>
```

Examples

Request

```
<loggingSyncOcns
    cookie="<real_cookie>"
    inFromOrZero="0"
    inToOrZero="4567000"/>
```

Response

List of event IDs.

methodVessel

This example shows a batch event notification that contains multiple configMoChangeEvent in a single payload.

Request Syntax

```
<xs:element name="methodVessel" type="methodVessel" substitutionGroup="externalMethod"/>
        <xs:complexType name="methodVessel" mixed="true">
            <xs:all>
                <xs:element name="inStimuli" type="MethodSet" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
        </xs:complexType>
```

Response Syntax

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

```
<methodVessel
    cookie=""
    commCookie=""
    srcExtSys="10.193.77.66"
    destExtSys="10.193.77.66"
    srcSvc="resource-mgr_dme"
    destSvc="sam_extXMLApi">
    <inStimuli>
        <configMoChangeEvent
            cookie=""
            commCookie=""
            srcExtSys="10.193.77.66"
            destExtSys="10.193.77.66"
            srcSvc="resource-mgr_dme"
            destSvc="sam_extXMLApi"
            inEid="434920">
        <inConfig>
            <resstateNormalizedHealthState
                dn="fw/inst-1010/normalized-health-state"
                operState="config"
                status="modified"/>
        </inConfig>
    </configMoChangeEvent>
```

```
    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434921">
        <inConfig>
            <fwComputeFirewall
                configState="applying"
                dn="org-root/org-tenant_d3337/org-dc1/cfw-vsn-d3340"
                status="modified"/>
        </inConfig>
    </configMoChangeEvent>

    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434922">
        <inConfig>
            <faultInst
                dn="org-root/org-tenant_d3337/org-dc1/cfw-vsn-d3340/fault-F0117"
                lc="flapping,soaking-clear"
                status="modified"/>
        </inConfig>
    </configMoChangeEvent>
    </inStimuli>
</methodVessel>
Event (10:32:2:807):

<methodVessel
cookie=""
commCookie=""
srcExtSys="10.193.77.66"
destExtSys="10.193.77.66"
srcSvc="resource-mgr_dme"
destSvc="sam_extXMLApi">
  <inStimuli>

    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434923">
    <inConfig>
        <eventLog
            dn="event-log"
            size="1412"
            status="modified"/>
    </inConfig>
    </configMoChangeEvent>

    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
```

```
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434924">
        <inConfig>
            <eventRecord
                affected="fw/inst-1009"
                cause="transition"
                changeSet=""
                code="E4194509"
                created="2010-11-19T18:32:02.622"
                descr="[FSM:STAGE:REMOTE-ERROR]: WRONG STATE:Result: not-applicable Code:
unspecified Message: (sam:dme:FwInstanceAssociate:configPA)"
                dn="event-log/83081"
                id="83081"
                ind="state-transition"
                severity="info"
                status="created"
                trig="special"
                txId="219376"
                user="internal"/>
        </inConfig>
    </configMoChangeEvent>
    </inStimuli>
</methodVessel>
```

Response

```
<methodVessel
    cookie=""
    commCookie=""
    srcExtSys="10.193.77.66"
    destExtSys="10.193.77.66"
    srcSvc="resource-mgr_dme"
    destSvc="sam_extXMLApi">
    <inStimuli>
        <configMoChangeEvent
            cookie=""
            commCookie=""
            srcExtSys="10.193.77.66"
            destExtSys="10.193.77.66"
            srcSvc="resource-mgr_dme"
            destSvc="sam_extXMLApi"
            inEid="434925">
            <inConfig>
                <resstateNormalizedHealthState
                dn="fw/inst-1010/normalized-health-state"
                operState="config-failure"
                status="modified"/>
            </inConfig>
        </configMoChangeEvent>

    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434926">
        <inConfig>
            <fwComputeFirewall
            configState="failed-to-apply"
            dn="org-root/org-tenant_d3337/org-dc1/cfw-vsn-d3340"
```

```
            status="modified"/>
        </inConfig>
    </configMoChangeEvent>

    <configMoChangeEvent
        cookie=""
        commCookie=""
        srcExtSys="10.193.77.66"
        destExtSys="10.193.77.66"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="434927">
        <inConfig>
            <faultInst
            dn="org-root/org-tenant_d3337/org-dc1/cfw-vsn-d3340/fault-F0117"
            lastTransition="2010-11-19T18:32:02.680"
            lc="flapping"
            status="modified"/>
        </inConfig>
    </configMoChangeEvent>
    </inStimuli>
</methodVessel>
```

orgResolveElements

Within a specified DN, this example retrieves managed objects that satisfy a query filter and searches managed objects starting at an organization, and optionally in the child organizations.

If there is no organization with that DN, an empty map is returned. If found, it searches managed objects with specified class and filters.

If *inHierarchical* = true, all matching objects and descendants are returned. If false, it returns only matching objects. If *inSingleLevel* = true, search stops at the organization level. If false, includes child organizations.

Request Syntax

```
<xs:element name="orgResolveElements" type="orgResolveElements"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveElements" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inClass" type="namingClassId"/>
            <xs:attribute name="inSingleLevel">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
```

```
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="no"/>
                                    <xs:enumeration value="yes"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:union>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="cookie" type="xs:string"/>
                <xs:attribute name="response" type="YesOrNo"/>
                <xs:attribute name="dn" type="referenceObject"/>
            </xs:complexType>
```

Response Syntax

```
<xs:element name="orgResolveElements" type="orgResolveElements"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveElements" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_5">
                        <xs:selector xpath="pair"/>
                        <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </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

```
<orgResolveElements
    dn="org-root/org-Cola"
    cookie="<real_cookie>"
    commCookie="7/15/0/19"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    inClass="policyPolicySet"
    inSingleLevel="no"
    inHierarchical="no">
    <inFilter>
    </inFilter>
</orgResolveElements>
```

Response

```
<orgResolveElements
    dn="org-root/org-Cola"
    cookie="<real_cookie>"
    commCookie="7/15/0/19"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
```

```
    destSvc="policy-mgr_dme"
    response="yes"
    errorCode="0"
    errorDescr="">
    <outConfigs>
        <pair key="pset-default">
            <policyPolicySet
            adminState="enabled"
            descr="The default Policy Set"
            dn="org-root/pset-default"
            intId="10082"
            name="default"/>
        </pair>
        <pair key="pset-myPolicySet3">
            <policyPolicySet
            adminState="enabled"
            descr=""
            dn="org-root/org-Cola/pset-myPolicySet3"
            intId="12289"
            name="myPolicySet3"/>
        </pair>
        <pair key="pset-policySetSanity">
            <policyPolicySet
            adminState="enabled"
            descr=""
            dn="org-root/org-Cola/pset-policySetSanity"
            intId="24627"
            name="policySetSanity"/>
        </pair>
        <pair key="pset-pci_compliance_f">
            <policyPolicySet
            adminState="enabled"
            descr=""
            dn="org-root/pset-pci_compliance_f"
            intId="24539"
            name="pci_compliance_f"/>
        </pair>
        <pair key="pset-pci_compliance_h">
            <policyPolicySet
            adminState="enabled"
            descr=""
            dn="org-root/pset-pci_compliance_h"
            intId="24541"
            name="pci_compliance_h"/>
        </pair>
    </outConfigs>
</orgResolveElements>
```

orgResolveInScope

This example shows how the system looks up the organization with the given DN, and (optional) parent organizations recursively to the root. If no organization, an empty map is returned. If found, it searches all pools with specified class and filters.

Note If *inSingleLevel* = false, the system searches parent organizations up to the root directory.

Request Syntax

```
<xs:element name="orgResolveInScope" type="orgResolveInScope"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveInScope" mixed="true">
            <xs:all>
                <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="inClass" type="namingClassId"/>
            <xs:attribute name="inSingleLevel">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="no"/>
                                <xs:enumeration value="yes"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="orgResolveInScope" type="orgResolveInScope"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveInScope" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_6">
                        <xs:selector xpath="pair"/>
                        <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </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

```
<orgResolveInScope
    cookie="<real_cookie>"
    dn="org-root/org-Cola"
    inClass="policyVirtualNetworkServiceProfile"
    inHierarchical="true"
    InSingleLevel="false" >
    <inFilter>
        <eq class="policyVirtualNetworkServiceProfile"
        property="name"
        value="spsanity" />
    </inFilter>
</orgResolveInScope>
```

Response

```
<orgResolveInScope
    dn="org-root/org-Cola"
    cookie="<real_cookie>"
    commCookie="7/15/0/1c35"
    srcExtSys="10.193.34.70"
    destExtSys="10.193.34.70"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="vnsp-spsanity">
            <policyVirtualNetworkServiceProfile
                childAction="deleteNonPresent"
                descr=""
                dn="org-root/org-Cola/vnsp-spsanity"
                intId="82018"
                name="spsanity"
                policySetNameRef="policySetSanity"
                vnspId="41">
            <policyVnspAVPair
                childAction="deleteNonPresent"
                descr=""
                id="1"
                intId="82019"
                name=""
                rn="vnsp-avp-1">
                <policyAttributeValue
                    childAction="deleteNonPresent"
                    id="1"
                    rn="attr-val1"
                    value="DEV"/>
                <policyAttributeDesignator
                    attrName="dept"
                    childAction="deleteNonPresent"
                    rn="attr-ref"/>
                </policyVnspAVPair>
            </policyVirtualNetworkServiceProfile>
        </pair>
    </outConfigs>
</orgResolveInScope>
```

orgResolveLogicalParents

This example shows how the system looks up logical parents with the specified DN up to the root directory.

Request Syntax

```
<xs:element name="orgResolveLogicalParents" type="orgResolveLogicalParents"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveLogicalParents" mixed="true">
            <xs:attribute name="inSingleLevel">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="inHierarchical">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>
            <xs:attribute name="dn" type="referenceObject"/>
        </xs:complexType>
```

Response Syntax

```
<xs:element name="orgResolveLogicalParents" type="orgResolveLogicalParents"
substitutionGroup="externalMethod"/>
        <xs:complexType name="orgResolveLogicalParents" mixed="true">
            <xs:all>
                <xs:element name="outConfigs" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_7">
                         <xs:selector xpath="pair"/>
                         <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </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

```
<orgResolveLogicalParents
    dn="org-root/org-Cisco/org-HR/zone-clients"
    cookie="<real_cookie>"
    commCookie="7/15/0/12"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    inSingleLevel="no"
    inHierarchical="no">
</orgResolveLogicalParents>
```

Response

```
<orgResolveLogicalParents
    dn="org-root/org-Cisco/org-HR/zone-clients"
    cookie="<real_cookie>"
    commCookie="7/15/0/12"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="policy-mgr_dme"
    response="yes"
    errorCode="0"
    errorDescr="">
    <outConfigs>
        <pair key="org-root/org-Cisco/zone-clients">
            <policyZone
                descr=""
                dn="org-root/org-Cisco/zone-clients"
                intId="24841"
                name="clients"/>
        </pair>
    </outConfigs>
</orgResolveLogicalParents>
```

policyEstimateImpact

This example takes a managed object (MO) and returns a list of MOs impacted by this change. Use it to estimate any impacts of a policy change (created, modified, or deleted).

Policy changes that can impact the MOs include the following:

- Modifying a name reference (that is, the resolved name is changed)
- On a deletion, another object reference referring to this object may now resolve to a different object with the same name higher up in the tree or may resolve to a default object.
- On creation, an object reference can refer to a newly created object. In the response, the outDeleteAllowed parameter indicates if the specified object can be deleted and, in such a case, potential name references referring to this object refer to another object higher up in the tree.
- The specified object cannot be deleted if it is a referred object in an object reference and there is no other object in the tree that can override this.

Request Syntax

```
<!--
           - Method: policy:EstimateImpact
         -->
        <xs:element name="policyEstimateImpact" type="policyEstimateImpact"
substitutionGroup="externalMethod"/>

        <xs:complexType name="policyEstimateImpact" mixed="true">
            <xs:all>
                <xs:element name="inConfig" type="configConfig" minOccurs="0"/>
            </xs:all>

            <xs:attribute name="cookie" type="xs:string"/>
            <xs:attribute name="response" type="YesOrNo"/>

        </xs:complexType>
```

Response Syntax

```
<!--
           - Method: policyEstimateImpact
         -->
        <xs:element name="policyEstimateImpact" type="policyEstimateImpact"
substitutionGroup="externalMethod"/>

        <xs:complexType name="policyEstimateImpact" mixed="true">
            <xs:all>
                <xs:element name="outImpactedMoSet" type="configMap" minOccurs="0">
                    <xs:unique name="unique_map_key_8">
                         <xs:selector xpath="pair"/>
                         <xs:field xpath="@key"/>
                    </xs:unique>
                </xs:element>
            </xs:all>

            <xs:attribute name="outDeleteAllowed">
                <xs:simpleType>
                    <xs:union memberTypes="xs:boolean">
                         <xs:simpleType>
                             <xs:restriction base="xs:string">
                                 <xs:enumeration value="no"/>
                                 <xs:enumeration value="yes"/>
                             </xs:restriction>
                         </xs:simpleType>
                    </xs:union>
                </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

```
<policyEstimateImpact
   cookie="1309385844/5d9899e6-5b5b-4ab0-b12c-cb6e50b5a7db"
   commCookie="7/12/0/1d18"
   srcExtSys="10.193.33.221"
   destExtSys="10.193.33.221"
   srcSvc="sam_extXMLApi"
   destSvc="policy-mgr_dme">
     <inConfig>
     <policyRuleBasedPolicy
     dn="org-root/org-tenant0/pset-myPolicySet1"
     status="deleted,modified"/>
     </inConfig>
   </policyEstimateImpact>
```

Response

```
<policyEstimateImpact
 cookie="1309385844/5d9899e6-5b5b-4ab0-b12c-cb6e50b5a7db"
 commCookie="7/12/0/1d18"
 srcExtSys="10.193.33.221"
 destExtSys="10.193.33.221"
 srcSvc="sam_extXMLApi"
 destSvc="policy-mgr_dme"
 response="yes"
 errorCode="0"
 errorDescr=""
 outDeleteAllowed="yes">
   <outImpactedMoSet>
 <pair key="org-root/org-tenant0/vnsp-VNSP1">
   <policyVirtualNetworkServiceProfile
   descr=""
   dn="org-root/org-tenant0/vnsp-VNSP1"
   intId="40930"
   name="VNSP1"
   policySetNameRef="myPolicySet1"
   vnspId="17"/>
 </pair>
   </outImpactedMoSet>
 </policyEstimateImpact>
```

poolResolveInScope

This example shows how the system looks up the pool with the given DN,, and (optional) parent pools recursively to the root. If no pool exists, an empty map is returned. If found, the system searches all pools with the specified class and filters.

Note If *inSingleLevel* = false, the system searches parent pools up to the root directory.

Request Syntax

```
<xs:element name="poolResolveInScope" type="poolResolveInScope"
substitutionGroup="externalMethod"/>
        <xs:complexType name="poolResolveInScope" mixed="true">
            <xs:all>
```

```
            <xs:element name="inFilter" type="filterFilter" minOccurs="0"/>
        </xs:all>
        <xs:attribute name="inClass" type="namingClassId"/>
        <xs:attribute name="inSingleLevel">
            <xs:simpleType>
                <xs:union memberTypes="xs:boolean">
                     <xs:simpleType>
                         <xs:restriction base="xs:string">
                             <xs:enumeration value="no"/>
                             <xs:enumeration value="yes"/>
                         </xs:restriction>
                     </xs:simpleType>
                </xs:union>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="inHierarchical">
            <xs:simpleType>
                <xs:union memberTypes="xs:boolean">
                     <xs:simpleType>
                         <xs:restriction base="xs:string">
                             <xs:enumeration value="no"/>
                             <xs:enumeration value="yes"/>
                         </xs:restriction>
                     </xs:simpleType>
                </xs:union>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="cookie" type="xs:string"/>
        <xs:attribute name="response" type="YesOrNo"/>
        <xs:attribute name="dn" type="referenceObject"/>
    </xs:complexType>
```

Response Syntax

```
<xs:element name="poolResolveInScope" type="poolResolveInScope"
substitutionGroup="externalMethod"/>
    <xs:complexType name="poolResolveInScope" mixed="true">
        <xs:all>
            <xs:element name="outConfigs" type="configMap" minOccurs="0">
                <xs:unique name="unique_map_key_9">
                     <xs:selector xpath="pair"/>
                     <xs:field xpath="@key"/>
                </xs:unique>
            </xs:element>
        </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

```
<poolResolveInScope
    dn="org-root/org-Cisco"
    cookie="<real_cookie>"
```

```
    class=fwPool />
```

Response

```
<poolResolveInScope
    dn="org-root/org-Cisco"
    cookie="<real_cookie>"
    commCookie="5/15/0/5bf"
    srcExtSys="10.193.33.221"
    destExtSys="10.193.33.221"
    srcSvc="sam_extXMLApi"
    destSvc="resource-mgr_dme"
    response="yes">
    <outConfigs>
        <pair key="fwpool-default">
            <fwPool
                assigned="0"
                descr="Default Pool of Virtual Security Gateway resources"
                dn="org-root/fwpool-default"
                fltAggr="65536"
                id="1"
                intId="10065"
                name="default"
                size="0"/>
        </pair>
</poolResolveInScope>
```

INDEX