

Cisco Virtual Network Management Center 2.1 XML API Guide



June 3, 2013

Contents

INTRODUCING THE CISCO VNMC XML API.....	6
OVERVIEW OF VNMC AND XML APIS	6
<i>VNMC.....</i>	<i>6</i>
<i>VNMC Management Information Model.....</i>	<i>6</i>
<i>VNMC Components.....</i>	<i>7</i>
Management Controller.....	7
Service Registry	7
Resource Manager	7
Policy Manager.....	8
VM Manager	8
<i>VNMC XML API.....</i>	<i>8</i>
<i>VNMC Data Model Schema</i>	<i>9</i>
<i>Accessing VNMC Services.....</i>	<i>10</i>
<i>Object Naming</i>	<i>10</i>
Distinguished Name	10
Relative Name.....	10
API METHOD CATEGORIES	10
<i>Authentication Methods</i>	<i>11</i>
<i>Query Methods</i>	<i>11</i>
<i>Query Filters.....</i>	<i>11</i>
Simple Filters.....	12
Property Filters.....	12
Composite Filters	12
Modifier Filters.....	13
<i>Configuration Methods.....</i>	<i>13</i>
DATA VALIDATION	13
EVENT SUBSCRIPTION METHODS	13
<i>Subscribing to Event Notification.....</i>	<i>14</i>
configMoChangeEvent	14
methodVessel.....	14

UML DIAGRAMS.....	15
<i>Classes.....</i>	15
<i>Associations</i>	16
<i>Inheritance.....</i>	17
<i>Aggregations and Compositions.....</i>	17
CAPTURING XML INTERCHANGE BETWEEN THE VNMC GUI AND SERVER.....	18
SUCCESS OR FAILURE RESPONSES	18
<i>Successful Responses</i>	18
<i>Failure Responses.....</i>	19
<i>Empty Result Responses.....</i>	19
COMMON API METHODS AND CONVENTIONS.....	20
METHODS AND FILTERS.....	20
<i>Authentication Methods</i>	20
Login.....	20
Refreshing the Session	21
Logging Out of the Session	21
Response to a Failed Login	21
<i>Query Methods for Information Gathering.....</i>	21
configResolveDn.....	21
configResolveDns	22
configResolveClass	22
configResolveClasses.....	22
configFindDnsByClassId.....	22
configResolveChildren.....	22
configResolveParent.....	23
configScope	23
QUERY METHODS FOR POLICIES.....	23
QUERY METHODS FOR FAULTS.....	24
FILTERS.....	24
<i>Simple Filters.....</i>	25
False Conditions	25
True Conditions	25
<i>Property Filters.....</i>	25
Equality Filter	25
Not Equal Filter	26
Greater Than Filter	26
Greater Than or Equal to Filter.....	26
Less Than Filter.....	26
Less Than or Equal to Filter	27
Wildcard Filter	27
Any Bits Filter	28
All Bits Filter	28
<i>Composite Filters</i>	29
And Filter.....	29
Or Filter	29
Between Filter	29
And Or Not Composite Filters	30
<i>Modifier Filters.....</i>	30
Not Filter	30

API EXAMPLES.....	31
<i>Authentication Using the Management Controller.....</i>	31
Authentication Request	31
Authentication Response	31
<i>Tenant Management Using the Service Registry.....</i>	31
Create or Update Organization Request	32
Create or Update Organization Response.....	32
POLICY MANAGEMENT	32
<i>Device Policies.....</i>	33
Syslog Policy	33
SNMP Policy	34
LogProfile Policy.....	35
Device Profiles.....	35
Zone	37
Object Group.....	38
Attribute Dictionary	39
Policy.....	39
PolicySet.....	41
Compute Security Profile	42
RESOURCE MANAGEMENT	43
<i>Create an Edge Firewall.....</i>	43
<i>Create a Compute Firewall.....</i>	45
<i>Assign a Device Profile to a Compute Firewall.....</i>	46
<i>Query Firewall Instances.....</i>	46
VNMC XML API METHODS.....	48
UNSUPPORTED METHODS.....	48
SUPPORTED METHODS	48
<i>aaaGetRemoteUserRoles.....</i>	48
Request Syntax	48
Response Syntax	49
Example.....	49
<i>aaaGetUserLocales</i>	49
Request Syntax.....	49
Response Syntax	50
Example.....	50
<i>aaaKeepAlive</i>	51
Request Syntax.....	51
Response Syntax	51
Example.....	51
<i>aaaLogin</i>	51
Request Syntax.....	51
Response Syntax	52
Example.....	53
<i>aaaLogout.....</i>	53
Request Syntax.....	53
Response Syntax	53
Example.....	54
<i>aaaRefresh.....</i>	54
Request Syntax.....	54
Response Syntax	54
Example.....	55
<i>configConfFiltered.....</i>	56
Request Syntax.....	57
Response Syntax	57

Example.....	58
<i>configConfMo</i>	58
Request Syntax	58
Response Syntax	59
Example.....	59
<i>configConfMoGroup</i>	60
Request Syntax	60
Response Syntax	60
Example.....	61
<i>configConfMos</i>	61
Request Syntax	62
Response Syntax	62
Example.....	62
<i>configFindDnsByClassId</i>	64
Request Syntax.....	64
Response Syntax	64
Example.....	64
<i>configResolveChildren</i>	65
Request Syntax	65
Response Syntax	65
Example.....	65
<i>configResolveClass</i>	66
Request Syntax.....	66
Response Syntax	67
Example.....	67
<i>configResolveClasses</i>	67
Request Syntax	68
Response Syntax	68
Example.....	68
<i>configResolveDn</i>	69
Request Syntax.....	69
Response Syntax	70
Example.....	70
<i>configResolveDns</i>	71
Request Syntax	71
Response Syntax	71
Example.....	71
<i>configResolveParent</i>	72
Request Syntax	72
Response Syntax	72
Example.....	73
<i>configScope</i>	73
Request Syntax	73
Response Syntax	74
Example.....	74
<i>eventSendHeartbeat</i>	75
Request Syntax.....	75
Response Syntax	75
Example.....	75
<i>eventSubscribe</i>	75
Request Syntax	75
Response Syntax	76
Example.....	76
<i>eventSubscribeApps</i>	76
Request Syntax	76
Response Syntax	76
Example.....	76

<i>faultAckFault</i>	77
Request Syntax	77
Response Syntax	77
Example.....	77
<i>faultAckFaults</i>	78
Request Syntax.....	78
Response Syntax	78
Example.....	78
<i>faultResolveFault</i>	78
Request Syntax.....	79
Response Syntax	79
Example.....	79
<i>loggingSyncOcn</i> s.....	80
Request Syntax.....	80
Response Syntax	80
Example.....	80
<i>orgResolveElements</i>	80
Request Syntax.....	80
Response Syntax	81
Example.....	81
<i>orgResolveInScope</i>	83
Request Syntax.....	83
Response Syntax	83
Example.....	84
<i>poolResolveInScope</i>	85
Request Syntax.....	85
Response Syntax	85
Example.....	86
APPENDIX: UML DIAGRAMS.....	87
<i>VPN Model</i>	87
<i>Generic Rule-Based Policy Model</i>	88

Introducing the Cisco VNMC XML API

The following sections provide general information about Cisco Virtual Network Management Center (VNMC) and the XML application programming interface (API):

- [Overview of VNMC and the XML API](#)
- [API Method Categories](#)
- [Data Validation](#)
- [Event Subscription Methods](#)
- [UML Diagrams](#)
- [Capturing XML Interchange Between the VNMC GUI and Server](#)
- [Success or Failure Responses](#)

Overview of VNMC and the XML API

The following sections provide an overview of VNMC and the XML API:

- [VNMC](#)
- [VNMC Management Information Model](#)
- [VNMC Components](#)
- [VNMC XML API](#)
- [VNMC Data Model Schema](#)
- [Accessing VNMC Services](#)
- [Object Naming](#)
- [Authentication Methods](#)

VNMC

VNMC is a virtual appliance that provides centralized device and security policy management for Cisco virtual services. Designed to support enterprise and multiple-tenant cloud deployments, VNMC provides transparent, seamless, and scalable management for securing virtualized data center and cloud environments.

With a built-in GUI and an XML API, VNMC enables you to configure, deploy, and manage virtual services throughout the data center from a central location.

VNMC is built on an information model-driven architecture in which each managed device is represented by its subcomponents (or objects), which are parametrically defined. This model-centric approach provides a flexible and simple mechanism for securing a virtualized infrastructure with compute and edge firewalls.

VNMC supports multiple client organizations, or tenants, that have their own virtualized compute, network, and storage resources, and that are deployed across a shared physical infrastructure. Multiple tenants can coexist on the same infrastructure, with each tenant maintaining administrative privileges for its virtualized resources. This multiple-tenancy design enables you to meet the specified service level agreement (SLA) for each tenant, including compute, network, storage, and security policies.

VNMC Management Information Model

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

Managed objects are abstractions of VNMC-managed entities, such as policies, rules, security profiles, and compute and edge firewalls. By invoking the API, objects are read from and written to the management information tree. The information model of an individual VNMC service component is centrally stored and managed by the Data Management Engine (DME). When a user initiates an administrative change to a VNMC service component (for example, by associating an edge firewall profile to an ASA 1000V, the DME first applies that change to the information model, and later the change is applied to the actual ASA 1000V. This approach is called a *model-driven framework*.

VNMC Components

VNMC consists of multiple service components that provide modularized functions for management, tenant management, policy management, resource management, and so on. These components are also called service providers or applications. Each component is accessed through a unique URL.

The following sections describe the VNMC components:

- [Management Controller](#)
- [Service Registry](#)
- [Resource Manager](#)
- [Policy Manager](#)
- [VM Manager](#)

Each component has its own data model and a DME that processes the model-driven service requests. Each component maintains its own management information tree storage (both in memory and in the persistent storage). Data sharing across different service components is achieved with ad-hoc interservice API communications or by using a publish or subscribe method.

Management Controller

Management Controller, also known as the core service, provides system-related services for the VNMC virtual machine, such as:

- Authenticates and authorizes user logins in local or LDAP mode.
- Provides access control, such as locales, roles, and trusted points.
- Maintains system information, such as the IP address, subnet mask, gateway, and hostname.
- Upon user input, performs system maintenance operations, 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 requests related to the Management Controller.

Service Registry

Service Registry is the central service repository that holds information about all registered managed endpoints (such as ASA 1000Vs or VSGs) and the service providers (such as Policy Manager or Resource Manager).

Note Service endpoints are referred to as clients in the GUI.

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

- Upon user input, creates, deletes, and updates organizations (tenants, data centers, applications, and tiers). Organization changes are automatically propagated to Policy Manager and Resource Manager when policies and resources are attached to the intended organizations.
- Maintains information about the registered services (such as providers, endpoints, and Management Controller).
- Maintains diagnostic information, such as audit logs, faults, and event logs.

The Service Registry type is service-reg. Use this service type in the API URL for all requests related to the Service Registry. Resource Manager manages logical compute and edge firewalls and their association with VSGs and

Resource Manager

Resource Manager manages logical compute and edge firewalls and their association with VSGs and ASA 1000Vs, respectively. When an edge firewall is associated with an ASA 1000V, the device configuration profile information (defined by the edge firewall) is pushed to the ASA 1000V which, in turn, triggers the ASA 1000V to download the security profiles and policies from Policy Manager. Resource Manager is responsible for the following services:

- Maintains an inventory of ASA 1000Vs, VSGs, and Cisco Virtual Supervisor Modules (VSMs).
- With user input, defines compute firewalls and associates them with VSGs for provisioning.

- With user input, defines edge firewalls and associates them with ASA 1000Vs for provisioning.
- Interacts with VMware vCenter instances to retrieve Virtual Machine (VM) attributes.
- Maintains an inventory of discovered VMs and distributes them to the VSGs with the following information:
 - VM attributes, such as name, hypervisor, parent vApp, and cluster
 - vNIC attributes, such as port profile name and IP address
- Manages pools of VSGs and ASA 1000Vs.
- Maintains health states and faults for VSGs and ASA 1000Vs.
- Maintains diagnostic information, such as audit logs, faults, and event logs.

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

Policy Manager

Policy Manager is the central repository for device configuration profiles, service policies, service profiles, and all associated artifacts. When a compute firewall is associated with a VSG from Resource Manager, the VSG queries Policy Manager to resolve the device profile, service profiles, and all referenced policies. The VSG then configures itself according to the information retrieved from Policy Manager. The same process is used when an edge firewall is associated with an ASA 1000V.

Policy Manager provides the following services:

- Using the user input, defines:
 - Firewall device profiles
 - Object groups
 - Policies
 - Policy rules with conditions on:
 - Network attributes such as protocol, source or destination, IP address, and port.
 - VM attributes such as instance name, guest OS, zone, parent application, port profile, cluster, resource pool, hostname, and hypervisor.
 - Custom attributes.
 - Policy sets
 - Security profile dictionary and custom attributes
 - Service profiles
 - VNMC system management device profiles and policies for NTP, DNS, syslog, and faults.
 - Virtual zones (vZones)
- Using the user input, assigns policies and policy sets.
- Distributes service policies, service profiles, device profiles, and associated objects to edge and compute firewall instances.
- Maintains diagnostic information such as audit logs, faults, and event logs.

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

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 service type is vm-mgr.

VNMC XML API

The VNMC XML API is a programmatic way of integrating and interacting with VNMC. 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, and is 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 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 DME. If multiple managed objects (such as policies) are being configured, and any of the managed objects cannot be configured, the API stops the 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* or *asynchronous*; that is, faster API processes are not blocked or delayed by those that require more time. A process receives a success message upon a valid request and a completion message when the task is finished.

Full event subscription is supported. VNMC sends notifications to all subscribers for the events that occur (such as changes to managed objects), 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.

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 (that is, configuration) of the endpoint.

Operation of the API is transactional and terminates on a single data model. VNMC is responsible for all endpoint (ASA 1000V, VSG, and VSM) communication (such as state updates). API users cannot communicate directly with endpoints, which relieves the users from administering isolated, individual component configurations.

The VNMC API model includes the following programmatic entities:

- **Classes**—Objects and their properties and states in the management information tree.
- **Methods**—Actions that the API performs on one or more objects.
- **Types**—Collections or ranges of allowed values for object properties. A typical request comes into a service component's DME and is placed in the transactor queue in first-in, first-out (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.

VNMC Data Model Schema

The data model HTML documents and schemas for all VNMC service providers are packaged with the VNMC server and can be accessed from either of the following URLs:

- <https://vnmc-ip-address/doc>, which includes:
 - core
 - policy-mgr
 - resource-mgr
 - service-reg
- <https://vnmc-ip-address/schema>, which 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 VNMC Services

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

```
https://vnmc-ip-address/xmlIM/service-type
```

where *service-type* is the type of the intended service provider as described in [VNMC Components](#). For example, to submit a request to the Policy Manager, use the service type `policy-mgr` as shown in the following example:

```
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) as described in the following sections:

- [Distinguished Name](#)
- [Relative Name](#)

Distinguished Name

A DN enables you to unambiguously identify a target object. DNs use the following format, which consists of a series of relative names:

```
dn = rn/rn/rn/rn...
```

For example, a series of relative names might resemble the following example:

```
org-root/org-tenant1/zone-trustedServers-0
```

In the following example, the DN provides a fully qualified path for a Zone (vZone in the GUI) 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-tenant1/zone-trustedServers-0" />
```

Relative Name

An RN identifies an object within the context of its parent object. The DN of an object consists of the parent DN and the RN in the following format:

```
dn = parent-dn/rn
```

For example, a Zone (vZone in the GUI) object with the name `trustedServers-0` that resides under the tenant `Tenant1` has the following related RN and DNs:

- The object's RN is `zone-trustedServers-0`.
- The object's parent tenant DN is `org-root/org-Tenant1`.
- The object's DN is `org-root/org-Tenant1/zone-trustedServers-0`.

API Method Categories

Four method categories are used to interact with VNMC. Each API is a method, and each method corresponds to an XML document. The following sections describe method categories in more detail and how to use query filters to retrieve the required information:

- [Authentication Methods](#)
- [Query Methods](#)
- [Query Filters](#)
- [Configuration Methods](#)
- [Event Subscription Methods](#)

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 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 initiate and maintain active VNMC sessions. 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.

A maximum of 256 sessions to VNMC can be opened at any one time. Subsequent login requests are rejected after the maximum session limit is reached.

Operations are performed by using the HTTP POST method. VNMC supports only the HTTPS protocol on port 443. The HTTP envelope contains the XML configuration.

[Table 1](#) describes the available authentication methods.

Table 1. Available Authentication Methods

Method	Description
aaaLogin	Initial method for logging into VNMC. Establishes a connection session and obtains a valid cookie.
aaaRefresh	Maintains the session and refreshes the current authentication cookie.
aaaLogout	Terminates the current session and deactivates the current authentication cookie.

Query Methods

Query methods obtain information on the current configuration state of a VNMC object.

Most query methods have the argument `inHierarchical`, with a Boolean value of `true/yes` or `false/no`. If true, the `inHierarchical` argument returns all child objects. For example:

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

API query methods can also include an `inRecursive` argument that specifies whether or not the call should be recursive; that is, whether the call should follow objects that point back to the referring objects or to the parent object.

[Table 2](#) describes the available query methods.

Table 2. Query Methods

Filter	Description
configResolveDns	Retrieves objects by a set of DNSs.
configResolveDns	Retrieves objects of multiple classes.
configResolveChildren	Retrieves the child objects of an object.
configResolveDns	Retrieves objects by a set of DNSs.
configResolveDns	Retrieves objects of multiple classes.
configResolveChildren	Retrieves the child objects of an object.
configResolveParent	Retrieves the parent object of an object.
configScope	Performs class queries on a DN in the management information tree.

Query Filters

The API includes filters that 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. The following sections describe the available filters:

- [Simple Filters](#)
- [Property Filters](#)
- [Composite Filters](#)
- [Modifier Filters](#)

Simple Filters

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

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

For more information, see [Simple Filters](#).

Property Filters

Property filters use object property values 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.

[Table 3](#) describes the types of property filters.

Table 3. Property Filters

Filter	Description
Equality	Result set contains objects with the identified property of equal to the provided property value.
Not equal	Result set contains objects with the identified property of not equal to the provided property value.
Greater	than Result set contains objects with the identified property of greater than the provided property value.
Greater than or equal	Result set contains objects with the identified property of greater than or equal to the provided property value.
Less than	Result set contains objects with the identified property of less than the provided property value.
Less than or equal	Result set contains objects with the identified property of less than or equal to the provided property value.
Wildcard	Result set contains objects with the identified property that satisfies the wildcard criteria. Supported wildcards are: %—Percent sign, matching a single character. *—Asterisk, matching zero or more characters. ?—Question mark, matching zero or one character. •- —Dash, indicating a range to match. For example, [a-e] would match any character between a and e. +—Plus sign, matching one or more characters.
Any bits	Result set contains objects with the identified property that has at least one of the passed bits set. (Available for use only on bitmask properties.)
All bits	Result set contains objects with the identified property that has all the passed bits set. (Available for use only on bitmask properties.)

For more information, see [Property Filters](#).

Composite Filters

Composite filters are composed of two or more component filters and thereby 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.

[Table 4](#) describes the available composite filters.

Table 4. Composite Filters

Filter	Description
AND	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.
Between	Result set contains those objects that fall between the range of the first specified value and second specified value. For example, all faults that
OR	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 the filter is composed of two equality filters
XOR	Result set contains those objects that pass the filtering criteria of no more than one of the composite's component filters.

For more information, see [Composite Filters](#).

Modifier Filters

Modifier filters change the results of a contained filter. Only the NOT modifier filter is supported. This filter negates the result of a contained filter. Use this filter to obtain objects that do not match contained criteria.

For more information, see [Modifier Filters](#).

Configuration Methods

Several methods enable you to make configuration changes to managed objects. Depending on the configuration method you use, the changes can be applied to the whole tree, a subtree, or an individual object.

Examples of configuration 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.

Data Validation

VNMC validates all configuration information that is entered via the API for each property, including the data type (such as integer, boolean, or string) and any additional constraints (such as an integer range or string regular expression). If the information supplied via the API does not meet the requirements for the specified property, VNMC stops the transaction and issues a failure response.

For more information about failure responses, see [Success or Failure Responses](#).

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 receive VNMC state change information by regular polling or subscribing to events. 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 VNMC. When an event occurs, VNMC sends a notification of the event and its type to the subscribing client applications. Only actual change events are sent, not the object's unaffected attributes. This process applies to all object changes in the system.

Subscribing to Event Notification

To subscribe to event notification:

- Step 1. Log into VNMC to obtain a valid cookie for event subscription. If you are not logged in, the event subscription request will be rejected with an error response.
- Step 2. Open an HTTP session and keep the session open.
- Step 3. Post the eventSubscribe request through the HTTP session as follows:

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

The eventSubscribe should be directed to the following URL:

```
https://<VNMCIP>/xmlIM/<DMEModule>
```

Depending on the module from which the client requires event notifications, the DME Module can be resource-mgr, policy-mgr, service-reg, or core. After the eventSubscribe request is accepted by VNMC, VNMC will send all new events as they occur through the HTTP session.

Each event has a unique event identifier (ID). 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, you can use loggingSyncOcn to retrieve the missed event.

The asynchronous event notifications for the following methods do not use the same format as other methods. Instead, event notifications for these methods use the format and syntax shown in the examples.

configMoChangeEvent

```
<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>
```

methodVessel

```
<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>
```

The following example shows how VNMC sends event notifications to the clients. The configMoChangeEvent is wrapped inside a methodVessel, which is used exclusively for an event notification.

```
<methodVessel cookie="<real_cookie>"
  commCookie=""
  srcExtSys="172.21.63.106"
  destExtSys="0.0.0.0"
  srcSvc="resource-mgr_dme"
  destSvc="sam_extXMLApi">
  <inStimuli>
    <configMoChangeEvent
      cookie=""
      commCookie=""
      srcExtSys="0.0.0.0"
```

```

        destExtSys="0.0.0.0"
        srcSvc="resource-mgr_dme"
        destSvc="sam_extXMLApi"
        inEid="52269">
        <inConfig>
            <vmInst
                dn="vmmEp/inst-5008b230-da65-54fb-76d4-8e1a462492cf"
                powerState="off"
                status="modified"/>
            </inConfig>
        </configMoChangeEvent>
    </inStimuli>
</methodVessel>

```

eventSendHeartbeat

VNMC sends no response or acknowledgement for the event Subscription events. Alternatively it sends an eventHeartBeat periodically (default 120 seconds) to the client conforming that the socket connection is not lost. If the client is receiving these heartbeats periodically, it can be assured that it is not missing any of the events in the VNMC. Here is an example of a eventHeartBeat that the VNMC sends to the clients.

```

<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>

```

UML Diagrams

Unified Modeling Language (UML) diagrams complement the object-oriented development approach used for VNMC. These diagrams display classes and associations between classes, along with attributes and methods, so that you can gain an overall view of VNMC classes and how the classes relate to one another.

The VNMC UML diagrams include the following attributes, as described in the subsequent topics:

- [Classes](#)
- [Associations](#)
- [Inheritance](#)
- [Aggregations and Compositions](#)

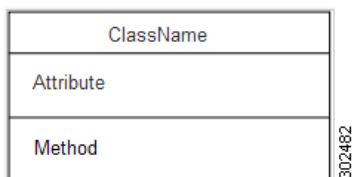
To view sample VNMC UML diagrams, see [UML Diagrams](#).

Classes

In UML, a *class* is a representation of an object. In VNMC, examples of objects, and therefore classes, are tenants, compute firewalls, edge firewalls, policies, and profiles. In general, anything that you can create in VNMC can be considered a class.

Classes are the primary components of UML and are presented as rectangles in UML diagrams, as shown in [Figure 1](#).

Figure 1 Representation of a Class in UML

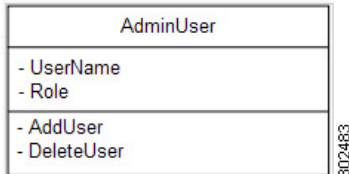


The class representation contains the following three sections:

- Class name—Name of the object.
- Attribute—Information that is stored about the object.
- Method—Operations that the object or class can perform.

[Figure 2](#) shows an example of a class with example entries in each section.

Figure 2 Example Class in UML



Not all class diagrams contain content in all sections. For example, you might see UML class diagrams with class name and attributes, but without methods. In these situations, the three sections are displayed, but the method section contains no information

Associations

An association in UML diagrams indicates that a link or dependency exists between two or more classes. Associations are represented in a number of ways, as shown in [Figure 3](#) and described in [Table 5](#).

Figure 3 Association in a UML Diagram

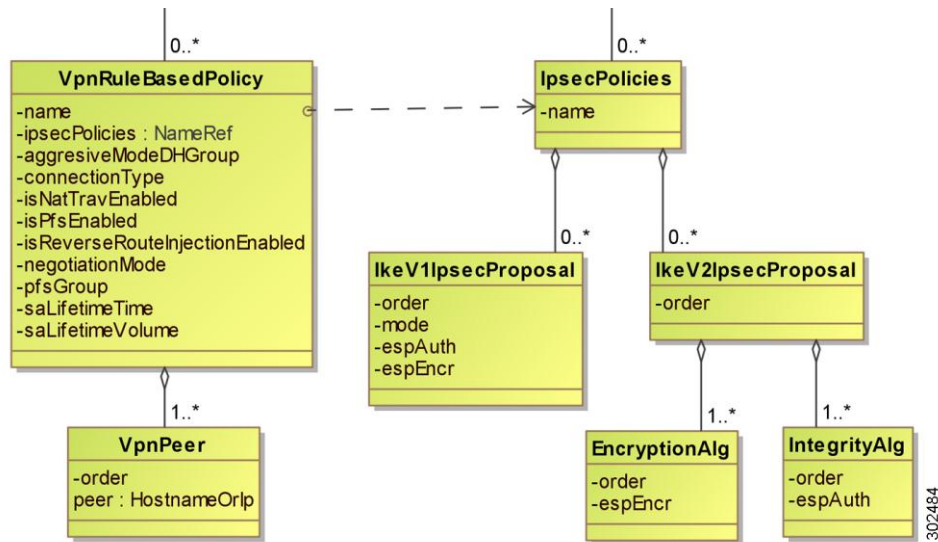


Table 5. UML Association Conventions

Convention	Description
Lines	
Solid line	<ul style="list-style-type: none"> • When used with a closed, unfilled arrowhead, indicates inheritance with the arrowhead pointing to the higher class object (or superclass). • When used with an open arrowhead, indicates an association to the known class.
Dashed line	Indicates a named reference, with the arrow originating from the referencing object.
Arrowheads	
Closed unfilled arrowhead	When used with a solid line, indicates inheritance, with the arrowhead pointing to the superclass.

Filled arrowhead	Indicates a nested flow of control or a procedure call.
Open arrowhead	When used with a solid line, indicates a unidirectional association with the arrowhead pointing to the known class.
Single arrowhead	<ul style="list-style-type: none"> When used with a solid line, indicates inheritance with the arrowhead pointing to the superclass. When used with a dashed line, indicates a unidirectional association, in the direction of the arrowhead.
Two or no arrowheads	When used with a solid line, indicates a bidirectional association.
Diamonds	
Empty diamond	Indicates a basic aggregation between objects with the empty diamond at the parent end.
Filled diamond	Indicates a composition aggregation between objects with the filled diamond at the parent end.
Multiplicity Indicators	
Indicate the number of instances of the object that can be associated with the connected object.	
0..1	Zero or one.
1	Only one.
0..*	Zero or more.
1..*	One or more.
n	Only n, where n > 1.
0..n	Only n, where n > 1.
1..n	Only n, where n > 1.

Inheritance

Similar or closely related classes often use many of the same attributes or methods. To prevent repetitive coding for multiple classes, UML employs an inheritance mechanism that enables you to reuse existing attributes and methods.

In an inheritance situation, the class that inherits the information is referred to as the subclass, and the class that is inherited from is referred to as the superclass. If all attributes and methods of one class are inherited by another class, that situation is referred to as pure inheritance.

In UML diagrams, inheritance is indicated by solid line with a closed arrowhead pointing from the subclass to the superclass.

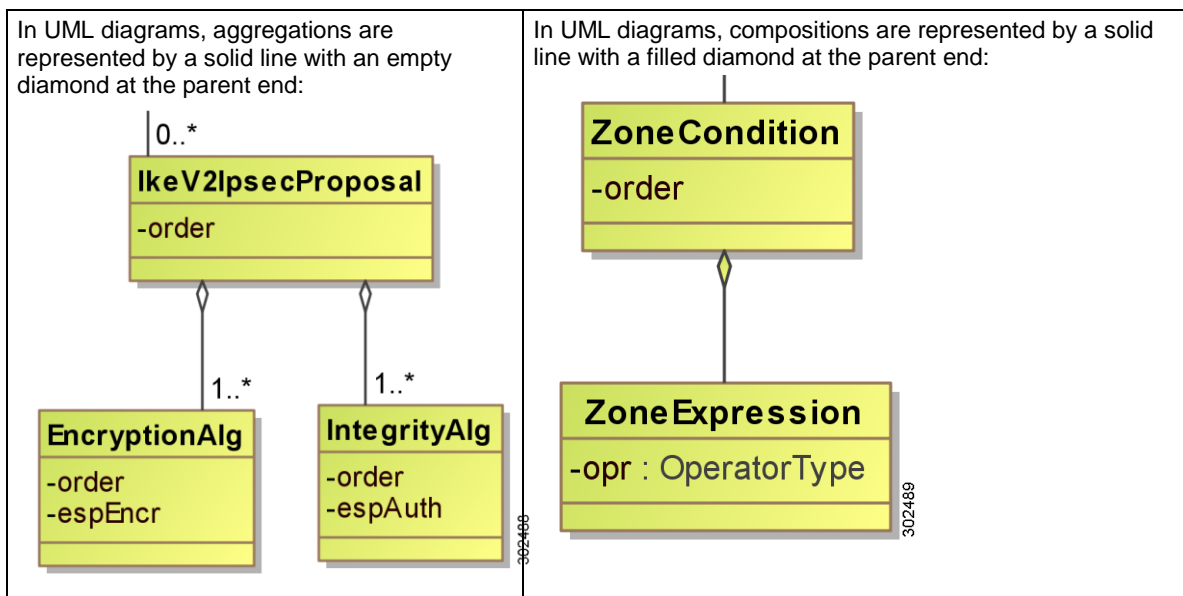
Aggregations and Compositions

Aggregations and compositions are types of associations that have unique characteristics. When working with UML diagrams, it is important to understand the differences between aggregations and compositions, and how they are presented in the diagrams.

[Table 6](#) describes the differences between aggregation and composition associations.

Table 6. Aggregation and Composition Characteristics

In an Aggregation..	In a Composition..
The aggregate object does not affect the existence of the component object. That is, if the aggregate object is removed from the system, the component objects continue to exist.	The aggregate object affects the existence of the component object. That is, if the aggregate object is removed from the system, all component objects are also removed.
A component object can be aggregated by other classes in the system.	A component object cannot be used by any object other than its aggregate (or parent) object.
Changes in the component object can be propagated to the rest of the system.	Changes in the component object cannot be propagated to the rest of the system.



Capturing XML Interchange Between the VNMC GUI and Server

The VNMC GUI is a web-based application that is developed using an Adobe Flex GUI framework. As a result, you can capture the XML interchange between the GUI and the VNMC server by installing a debug version of Adobe Flash Player. After you install the debug version of Adobe Flash Player, the log output is stored in a log file under your home directory. (In Windows 7, the log file can be found under `C:\Users\username\AppData\Roaming\Macromedia\Flash Player\Logs\flashlog.txt`.)

Most of the sample request and response content included in the Example sections in [VNMC XML API Methods](#) is captured in this manner.

Success or Failure Responses

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 indicates that the request is valid, but does not indicate that the operation was completed. For example, after a database backup request is accepted with a successful response from VNMC, the VNMC server might take a longer time to finish the actual backup.

The following sections describe response types in more detail:

- [Successful Responses](#)
- [Failure Responses](#)
- [Empty Result Responses](#)

Successful Responses

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

The following example shows a `configResolveDn` request on a policy with the DN `org-root/org-tenant d3337/pol-p1`:

```
<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>

```

Failure Responses

Responses to failed requests include XML attributes for `errorCode` and `errorDescr`. The following example shows the response to a failed request that tried to create a policy with the same name as another policy already 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>

```

For certain kind of parsing errors, if a property mentioned is not valid or unparseable, the response is an Error XML. The example below shows a response for an invalid Regular Expression used for a wildcard Filter.

```

<error cookie="<real_cookie>"
  response="yes"
  errorCode="ERR-xml-parse-error"
  invocationResult="594"
  errorDescr="XML PARSING ERROR: Invalid regular expression pattern"/>

```

Empty Result Responses

A query request for a nonexistent object is not treated as a failure. 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. The following 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>

```

Common API Methods and Conventions

The following sections identify common VNMC API methods, describe API conventions, and provide example requests and responses:

- [Methods and Filters](#)
- [API Examples](#)

Methods and Filters

The following topics describe the methods and filters that VNMC uses:

- [Authentication Methods](#)
- [Query Methods for Information Gathering](#)
- [Query Methods for Policies](#)
- [Query Methods for Faults](#)
- [Filters](#)

Authentication Methods

Authentication allows API interaction with 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.

The following topics describe authentication methods in more detail:

- [Login](#)
- [Refreshing the Session](#)
- [Logging Out of the Session](#)
- [Response to a Failed Login](#)

Login

The following code sample shows a login request using HTTPS to connect to VNMC, and a Linux POST command to post authentication requests.

```
POST https://10.193.34.70/xmlIM/mgmt-controller
Please enter content (application/x-www-form-urlencoded) to be POSTed:
<aaaLogin
  inName="admin"
  inPassword="Company@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, VNMC reads the request and performs the actions as provided in the method. VNMC responds with a message in XML document format and indicates success or failure of the request.

The following code sample shows a typical successful response for a login request.

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

```
</aaaLogin>
```

Note The VNMCM event channel uses HTTP, so it is neither encrypted nor 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.

The following code sample shows the method for refreshing a session.

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

Logging Out of the Session

The following code sample shows the method for logging out of a session.

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

Response to a Failed Login

The following code sample shows the response 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.

The following sections describe available query methods:

- [configResolveDn](#)
- [configResolveDns](#)
- [configResolveClass](#)
- [configResolveClasses](#)
- [configFindDnsByClassId](#)
- [configResolveChildren](#)
- [configResolveParent](#)
- [configScope](#)
- [configResolveDn](#)

configResolveDn

When using configResolveDn, note the following:

- The object specified by the DN is 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.

See the example request/response in [configResolveDn](#).

configResolveDns

When using configResolveDns to resolve multiple DNSs, note the following:

- The objects specified by the DNSs 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 DNSs are returned as part of the outUnresolved attribute.

See the example request/response in [configResolveDns](#).

configResolveClass

When using configResolveClass to resolve a class, note 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 get all instances of equipment, query the equipmentItem class. To obtain only a list of servers, use computeBlade as the attribute value for classId in the query.

See the example request/response in [configResolveClass](#).

configResolveClasses

When using configResolveClasses to resolve multiple classes, note 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 inIds attribute, an XML parsing error is generated. The query cannot execute.

See the example request/response in [configResolveClasses](#).

configFindDnsByClassId

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

- The DNSs 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](#).

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](#).

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](#).

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** If a class name 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](#).

Query Methods for Policies

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

The following code sample shows a query 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, as follows:

```

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

```

Query Methods for Faults

The following code sample shows a query for faults:

```

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

```

The following example, which contains the filter <inFilter> eq class= "faultInst", obtains a list of all major or critical faults:

```

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

```

Filters

Use filters to create specific and targeted queries, as described in the following topics:

- [Simple Filters](#)
- [Property Filters](#)

- [Composite Filters](#)
- [Modifier Filters](#)

Simple Filters

Simple filters use true and false conditions, as described in the following topics:

- [False Conditions](#)
- [True Conditions](#)

In addition to inHierarchical (shown in the following examples), inRecursive and inSingleLevel can also be set to true or false.

False Conditions

The following example shows a simple filter using a false condition:

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

True Conditions

The following example shows a simple filter using a true condition:

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

Property Filters

The following sections describe how to use property filters:

- [Equality Filter](#)
- [Not Equal Filter](#)
- [Greater Than Filter](#)
- [Greater Than or Equal to Filter](#)
- [Less Than Filter](#)
- [Less Than or Equal to Filter](#)
- [Wildcard Filter](#)
- [Any Bits Filter](#)
- [All Bits Filter](#)

Equality Filter

The following example shows an *equality* filter:

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

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

Not Equal Filter

The following 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

The following example shows a *greater than* filter:

```
<configResolveClass
  cookie="<real_cookie>"
  classId="eventRecord"
  inHierarchical="false">
  <inFilter>
    <gt class="eventRecord"
      property="txId"
      value="36520" />
  </inFilter>
</configResolveClass>
```

Greater Than or Equal to Filter

The following 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

The following 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

The following 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

The following example shows a *wildcard* filter that finds any virtual firewall whose name begins with the prefix "edge":

Request:

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

Response:

```
<configResolveClass
  cookie="<real_cookie>"
  commCookie="5/12/0/14dc"
  srcExtSys=""
  destExtSys=""
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes"
  classId="fwInstance">
  <outConfigs>
    <fwInstance
      assignedToDn="org-root/org-Tenant-1/efw-NewSampleEDgeFirewall"
      assoc="associated"
      capability="infra-fw"
      descr=""
      dn="fw/inst-1007"
      fltAggr="0"
      fsmDescr=""
      fsmPrev="AssociateSuccess"
      fsmProgr="100"
      fsmRmtInvErrCode="none"
      fsmRmtInvErrDescr=""
      fsmRmtInvRslt=""
      fsmStageDescr=""
      fsmStamp="2013-05-21T23:27:57.016"
      fsmStatus="nop"
      fsmTry="0"
      intId="10259"
      mgmtIp="<FirewallIP>"
```

```

        model=""
        name="edge-firewall"
        pooled="0"
        registeredClientDn="extpol/reg/clients/client-1007"
        revision="0"
        serial=""
        svcId="1007"
        vendor="" />
    </outConfigs>
</configResolveClass>

```

For wildcard filter, if the user provides an invalid filter, VNMC sends back a response in an error tag. Example below has a wildcard "*" which is not a valid wildcard filter.

Request

```

<configResolveClass
  cookie="<real_cookie>"
  inHierarchical="false"
  classId="orgTenant">
  <inFilter>
    <wcard class="orgTenant"
      property="name" value="*" />
  </inFilter>
</configResolveClass>

```

Response

```

<error cookie="<real_cookie>"
  response="yes"
  errorCode="ERR-xml-parse-error"
  invocationResult="594"
  errorDescr="XML PARSING ERROR: Invalid regular expression pattern"/>

```

Any Bits Filter

The following 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

The following 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

The following topics describe composite filters:

- [And Filter](#)
- [Or Filter](#)
- [Between Filter](#)
- [And Or Not Composite Filters](#)

And Filter

The following 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

The following 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

The following example shows a *between* filter:

```
<configResolveClass
  cookie="<real_cookie>"
  classId="eventRecord"
  inHierarchical="false">
  <inFilter>
    <bw class="eventRecord"
      property="txId"
```

```

        firstValue="4"
        secondValue="5"/>
    </inFilter>
</configResolveClass>

```

And Or Not Composite Filters

The following example shows an *and or not* composite filter that returns all objects of type `fwComputeFirewall` that have an associated state of `unassociated` or a config state of `not-applied`, but that do not have an auxiliary property value of `reachability`:

```

<configScope
  cookie="<real_cookie>"
  dn="org-root"
  inClass="fwComputeFirewall"
  inHierarchical="false"
  inRecursive="false">
  <inFilter>
    <and>
      <or>
        <eq class="fwComputeFirewall" property="assocState"
          value="unassociated" />
        <eq class="fwComputeFirewall" property="configState" value="not-
          applied" />
      </or>
      <not>
        <eq class="fwComputeFirewall" property="auxProps"
          value="reachability"/>
      </not>
    </and>
  </inFilter>
</configScope>

```

Modifier Filters

Not Filter

The following 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

The following sections provide API configuration and management examples:

- [Authentication Using the Management Controller](#)
- [Tenant Management Using the Service Registry](#)
- [Policy Management](#)
- [Resource Management](#)

The following sections demonstrate how configConfMos is used to create, modify, or delete an MO (or object). In each example where the configConfMos API uses a single XML element <pair> to specify a single configuration object, you can use the configConfMo API to obtain the same result.

Authentication Using the Management Controller

Access to 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, 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 VNMC.

The following topics provide examples of authentication requests and responses:

- [Authentication Request](#)
- [Authentication Response](#)

Authentication Request

The following 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

The following example 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 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 orgDatacenter—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 information:

- Object DN
- Attribute values
- Status equal to created or modified
- To delete an organization, use a status = deleted in the request.

The following topics provide examples of create or update requests and responses:

- [Create or Update Organization Request](#)
- [Create or Update Organization Response](#)

Create or Update Organization Request

The following example shows how to create a tenant named demoTenant:

```
<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

The following 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

The following topics describe how to manage policies:

- [Device Policies](#)
- [Device Profiles](#)
- [Zone](#)

- [Object Group](#)
- [Attribute Dictionary](#)
- [Policy](#)
- [PolicySet](#)
- [Compute Security Profile](#)

Device Policies

The following topics provide examples for working with device policies:

- [Syslog Policy](#)
- [SNMP Policy](#)
- [LogProfile Policy](#)

Syslog Policy

A syslog policy object tracks all actions that take place within the system and sets the logging level for a device profile. The following example creates a syslog policy named msyslog.

Request

```
<configConfMos
  cookie="<real_cookie>"
  <inConfigs>
    <pair key="org-root/syslog-msyslog">
      <commSyslog dn="org-root/syslog-msyslog">
        <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-msyslog">
      <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

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

Request

```

<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-23.host123.com"
        community="private"/>
      <commSnmpTrap
        hostname="nms-42.host123.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

The following 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-debugLog">
      <policyLogProfile
        dn="org-root/logprof-debugLog"
        name="debugLog"
        level="debug1"
        size="10000000"
        backupCount="4"/>
      </pair>
    </inConfigs>
  </configConfMos>

```

Response

```

<configConfMos
  cookie="<real_cookie>"
  commCookie="7/12/0/17d0"
  srcExtSys="172.20.101.150"
  destExtSys="172.20.101.150"
  srcSvc="sam_extXMLApi"
  destSvc="policy-mgr_dme"
  response="yes">
  <outConfigs>
    <pair key="org-root/logprof-debugLog">
      <policyLogProfile
        adminState="enabled"
        backupCount="4"
        descr=""
        dn="org-root/logprof-debugLog"
        intId="10514"
        level="debug1"
        name="debugLog"
        size="10000000"
        status="created"/>
      </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"
      snmpPolicy="mysnmp"
      syslogPolicy="mysyslog"
      timezone="America/Los_Angeles" >
    <commDns
      name="somedns.com"
      domain="somedns.com"/>
    <!-- The order attribute means the device should first use the DNS provider with
    thesmallest "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
      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

The following example creates two zones: trustedClients-0 and trustedServers-0. A set of attributes with values is also assigned.

Note In the VNMC UI, a zone is represented as a vZone (virtual Zone).

Request

POST URL: <https://10.193.33.221/xmlIM/policy-mgr>

XML API payload:

```

<configConfMos
  cookie="<real_cookie>"
  <inConfigs>
    <pair key="org-root/org-tenant1/zone-trustedClients-0">
<!-- Create a zone of all VMs in the 192.168.1.0/24 subnet -->
      <policyZone dn="org-root/org-tenant1/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-tenant1/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-tenant1/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-tenant1/zone-trustedServers-0">
        <policyZone
            descr=""
            dn="org-root/org-tenant1/zone-trustedServers-0"
            intId="24404"
            name="trustedServers-0"
            status="created"/>
        </pair>
    <pair key="org-root/org-tenant1/zone-trustedClients-0">
        <policyZone
            descr=""
            dn="org-root/org-tenant1/zone-trustedClients-0"
            intId="24408"
            name="trustedClients-0"
            status="created"/>
        </pair>
    </outConfigs>
</configConfMos>

```

Object Group

The following 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="7715/0/1a4"
    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

The following example creates a 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-tenant1 -->
        <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

Beginning with version 2.0, VNMC no longer uses the adminState property for ACL policies because VSG compute firewalls do not support a value of disabled for this property; the default value is enabled. However, VNMC continues to use the adminState property in other types of policies, such as those used by ASA 1000V.

If you set the `adminState` property via the API for an ACL policy, the response will contain the following error message:

```
<configConfMos
  cookie="<real_cookie>"
  commCookie="7/12/0/55"
  srcExtSys="10.193.76.15"
  destExtSys="10.193.76.15"
  srcSvc="sam_extXMLApi"
  destSvc="policy-mgr_dme"
  response="yes"
  errorCode="170"
  invocationResult="unidentified-fail"
  errorDescr="Admin implicit props cannot be modified, prop=adminState">
```

The following 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-tenant1/pol-trustedHosts">
      <policyRuleBasedPolicy dn="org-root/org-tenant1/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-tenant1/pol-trustedHosts">
    <policyRuleBasedPolicy
      descr=""
      dn="org-root/org-tenant1/pol-trustedHosts"
      intId="25131"
      name="trustedHosts"
      status="created"/>
    </pair>
  </outConfigs>
</configConfMos>

```

PolicySet

The following example creates ACL-PolicySet and sets the order in which policies are applied.

Request

```

<configConfMos
  cookie="<real_cookie>"
  inHierarchical="false">
  <inConfigs>
    <pair key="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test">
      <policyPolicyNameRef
        dn="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test"
        order="100"
        policyName="Test"
        status="created"/>
      </pair>
    <pair key="org-root/org-tenant1/pset-ACL-PolicySet">
      <policyPolicySet
        descr=""
        dn="org-root/org-tenant1/pset-ACL-PolicySet"
        name="ACL-PolicySet"
        status="created"/>
      </pair>
    <pair key="org-root/org-tenant/pset-ACL-PolicySet/polref-Test-03">
      <policyPolicyNameRef
        dn="org-root/org-tenant/pset-ACL-PolicySet/polref-Test-03"
        order="300"
        policyName="Test-03"
        status="created"/>
      </pair>
    <pair key="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-02">
      <policyPolicyNameRef
        dn="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-02"
        order="200"
        policyName="Test-02"
        status="created"/>
      </pair>
    </inConfigs>
  </configConfMos>

```

Response

```

<configConfMos
  cookie="<real_cookie>"

```

```

commCookie="7/12/0/187c"
srcExtSys="172.20.101.150"
destExtSys="172.20.101.150"
srcSvc="sam_extXMLApi"
destSvc="policy-mgr_dme"
response="yes">
<outConfigs>
  <pair key="org-root/org-tenant1/pset-ACL-PolicySet">
    <policyPolicySet
      adminState="enabled"
      descr="" dn="org-root/org-tenant1/pset-ACL-PolicySet"
      intId="10666"
      name="ACL-PolicySet"
      status="created"/>
  </pair>
  <pair key="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test">
    <policyPolicyNameRef
      dn="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test"
      order="100"
      policyName="Test"
      status="created"/>
  </pair>
  <pair key="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-02">
    <policyPolicyNameRef
      dn="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-02"
      order="200"
      policyName="Test-02"
      status="created"/>
  </pair>
  <pair key="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-03">
    <policyPolicyNameRef
      dn="org-root/org-tenant1/pset-ACL-PolicySet/polref-Test-03"
      order="300"
      policyName="Test-03"
      status="created"/>
  </pair>
</outConfigs>
</configConfMos>

```

Compute Security Profile

The following 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:

- Creates an edge firewall.
- Assigns compute and edge firewalls to policies.
- Queries for available firewall instances.
- Associates firewall instances with a managed object.
- Allows the binding of organizations to resource pools.
- Interacts with virtual centers to retrieve VM attributes.

The following topics provide examples for working with firewalls:

- [Create an Edge Firewall](#)
- [Create a Compute Firewall](#)
- [Assign a Device Profile to a Compute Firewall](#)
- [Query Firewall Instances](#)

Create an Edge Firewall

The following example creates an edge firewall with one inside data interface and one outside data interface.

Request

```
<configConfMos
  cookie="<real_cookie>"
  <inConfigs>
    <pair key="org-root/efw-default">
      <fwEdgeFirewall
        dn="org-root/efw-default"
        rn="efw-default"
        name="default"
        haMode="standalone"
        status="created"/>
    </pair>
    <pair key="org-root/efw-default/interface-inside">
```

```

    <fwDataInterface
      dn="org-root/efw-default/interface-inside"
      name="inside"
      role="inside"
      ipSubnet="255.255.255.0"
      ipAddressPrimary="10.10.10.1"
      status="created"/>
  </pair>
  <pair key="org-root/efw-default/interface-outside">
    <fwDataInterface
      dn="org-root/efw-default/interface-outside"
      name="outside"
      role="outside"
      ipSubnet="255.255.255.0"
      ipAddressPrimary="10.10.20.1"
      status="created"/>
  </pair>
</inConfigs>
</configConfMos>

```

Response

```

<configConfMos
  cookie="<real_cookie>"
  commCookie="5/12/0/35da"
  srcExtSys="172.20.101.150"
  destExtSys="172.20.101.150"
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes">
<outConfigs>
  <pair key="org-root/efw-default">
    <fwEdgeFirewall
      assocState="unassociated"
      auxProps=""
      configState="not-applied"
      descr=""
      dn="org-root/efw-default"
      fltAggr="0"
      haMode="standalone"
      hostname="edge-firewall"
      intId="10467"
      name="default"
      status="created"/>
  </pair>
  <pair key="org-root/efw-default/interface-inside">
    <fwDataInterface
      descr=""
      dn="org-root/efw-default/interface-inside"
      intId="10468"
      ipAddressPrimary="10.10.10.1"
      ipAddressSecondary="0.0.0.0"
      ipSubnet="255.255.255.0"
      isIpViaDHCP="no"
      name="inside"
      role="inside"
      status="created"/>
  </pair>
  <pair key="org-root/efw-default/interface-outside">
    <fwDataInterface
      descr=""
      dn="org-root/efw-default/interface-outside"

```

```

        intId="10469"
        ipAddressPrimary="10.10.20.1"
        ipAddressSecondary="0.0.0.0"
        ipSubnet="255.255.255.0"
        isIpViaDHCP="no"
        name="outside"
        role="outside"
        status="created"/>
    </pair>
</outConfigs>
</configConfMos>

```

Create a Compute Firewall

The following example creates a compute firewall with the name test5.

Request

```

<configConfMos
  cookie="<real_cookie>"
  inHierarchical="false">
  <inConfigs>
    <pair key="org-root/cfw-test5">
      <fwComputeFirewall
        dn="org-root/cfw-test5"
        name="test5"
        dataIpAddress="5.5.5.5"
        dataIpSubnet="255.255.255.0"
        status="created"/>
      </pair>
    </inConfigs>
  </configConfMos>

```

Response

```

<configConfMos
  cookie="<real_cookie>"
  commCookie="5/12/0/1545"
  srcExtSys="172.25.97.246"
  destExtSys="172.25.97.246"
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes">
  <outConfigs>
    <pair key="org-root/cfw-test5">
      <fwComputeFirewall
        assocState="unassociated"
        auxProps=""
        configState="not-applied"
        dataIpAddress="5.5.5.5"
        dataIpSubnet="255.255.255.0"
        descr=""
        devicePolicy=""
        dn="org-root/cfw-test5"
        fltAggr="0"
        hostname="firewall"
        intId="10583"
        name="test5"
        status="created"/>
      </pair>
    </outConfigs>
  </configConfMos>

```

Assign a Device Profile to a Compute Firewall

Beginning with VNMC 2.0, fw:ComputeFirewall.devicePolicy is deprecated and should not be modified via the XML API. Instead, create a child MO of type logical:DeviceProfileAssociation, and set logical:DeviceProfileAssociation.profileRef. The logical:DeviceProfileAssociation should be a child of fw:ComputeFirewall.

If you modify fw:ComputeFirewall.devicePolicy, you might see inconsistent behavior between fw:ComputeFirewall.devicePolicy and logical:DeviceProfileAssociation, and thus what is displayed in the UI.

The following example assigns the device profile my-profile-ref to the compute firewall test5, which was created in [Create a Compute Firewall, page 2-28](#).

Request

```
<configConfMos
  cookie="<real_cookie>"
  inHierarchical="false">
  <inConfigs>
    <pair key="org-root/cfw-test5/device-profile">
      <logicalDeviceProfileAssociation
        dn="org-root/cfw-test5/device-profile"
        profileRef="my-profile-ref"
        status="created"/>
    </pair>
  </inConfigs>
</configConfMos>
```

Response

```
<configConfMos
  cookie="<real_cookie>"
  commCookie="5/12/0/1571"
  srcExtSys="172.25.97.246"
  destExtSys="172.25.97.246"
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes">
<outConfigs>
  <pair key="org-root/cfw-test5/device-profile">
    <logicalDeviceProfileAssociation
      descr=""
      dn="org-root/cfw-test5/device-profile"
      intId="10586"
      name=""
      profileRef="my-profile-ref"
      status="created"/>
  </pair>
</outConfigs>
</configConfMos>
```

Query Firewall Instances

The following example queries all firewall instances with the management IP address of 10.193.33.221 and obtains 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>
```

VNMC XML API Methods

This section includes the following topics:

- [Unsupported Methods](#)
- [Supported Methods](#)

Unsupported Methods

Cisco does not support the following methods even though they appear in the Methods list on your VNMC server.

Caution: We strongly recommend that you do not use the following methods. They can cause unexpected results on your VNMC server.

- `aaaCheckComputeAuthToken`
- `aaaCheckComputeExtAccess`
- `aaaGetComputeAuthToken`
- `cliviewConfMos`
- `configFindDependencies`
- `configMoChangeEvent`
- `fsmDebugAction`
- `methodVessel`
- `orgResolveLogicalParents`
- `policyEstimateImpact`
- `statsClearInterval`
- `syntheticFSObjInventory`
- `syntheticTestTx`

Supported Methods

This section provides descriptions, request and response syntax, and usage examples for the following methods:

- [aaaGetRemoteUserRoles](#)
- [aaaGetUserLocales](#)
- [aaaKeepAlive](#)
- [aaaLogin](#)
- [aaaLogout](#)
- [aaaRefresh](#)
- [configConfFiltered](#)
- [configConfMo](#)
- [configConfMoGroup](#)
- [configConfMos](#)
- [configFindDnsByClassId](#)
- [configResolveChildren](#)
- [configResolveClass](#)
- [configResolveClasses](#)
- [configResolveDn](#)
- [configResolveDns](#)
- [configResolveParent](#)
- [configScope](#)
- [eventSendHeartbeat](#)
- [eventSubscribe](#)
- [eventSubscribeApps](#)
- [faultAckFault](#)
- [faultAckFaults](#)
- [faultResolveFault](#)
- [loggingSyncOcns](#)
- [orgResolveElements](#)
- [orgResolveInScope](#)
- [poolResolveInScope](#)

These methods are also called from the GUI console.

aaaGetRemoteUserRoles

This API method 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>

```

Example

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 API method 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>

```

Example

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

The following 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>

```

Example

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

The following example shows the login process that is required to begin a session and which establishes an authenticated HTTPS session between the client and 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: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>

```

Example

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

The following 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>

```

Example

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, 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>

```

Example

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>

```

configCloneMo

Certain Mos (for example, Orgs) can be cloned based on their hierarchy and the access permissions of the user who is performing the operation. If an Mo is cloned, all of its children are cloned as well with the default access permissions of the cloningMo. The syntax for this operation follows.

Request Syntax

```

<xs:element name="configCloneMo" type="configCloneMo"
substitutionGroup="externalMethod"/>
  <xs:complexType name="configCloneMo" mixed="true">
    <xs:attribute name="inToDn" type="referenceObject"/>
      <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="configCloneMo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="outConfig">
        <xs:complexType>
          <xs:sequence>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="dn"/>
      <xs:attribute type="xs:string" name="cookie"/>
      <xs:attribute type="xs:string" name="commCookie"/>
      <xs:attribute type="xs:string" name="srcExtSys"/>
      <xs:attribute type="xs:string" name="destExtSys"/>
      <xs:attribute type="xs:string" name="srcSvc"/>
      <xs:attribute type="xs:string" name="destSvc"/>
      <xs:attribute type="xs:string" name="response"/>
    </xs:complexType>
  </xs:element>

```

Example

Request

```

<configCloneMo cookie="<real_cookie>"
  dn="org-root/org-Tenant-1"
  inToDn="org-root/org-Tenant-3">
</configCloneMo>

```

Response

```

<configCloneMo
  dn="org-root/org-Tenant-1"
  cookie="<real_cookie> "
  commCookie="5/12/0/14e4"
  srcExtSys=""
  destExtSys=""
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes">
  <outConfig>

```



```

        <orgTenant
            childAction="deleteNonPresent"
            descr=""
            dn="org-root/org-Tenant-3"
            fltAggr="0"
            level="1"
            name="Tenant-3"
            status="created">
        </orgTenant>
    </outConfig>
</configCloneMo>

```

configConfFiltered

The following 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>

```

Example

Request

```
<configConfFiltered
  cookie="<real_cookie>"
  inHierarchical="false"
  classId="orgTenant">
  <inFilter>
    <eq class="orgTenant"
      property="name"
      value="Tenant1" />
  </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-tenant1/org-HR"
      fltAggr="0"
      level="2"
      name="HR"
      status="modified"/>
  </outConfigs>
</configConfFiltered>
```

configConfMo

The following 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:restriction>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
```

```

        <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>

```

Example

Request

```

<configConfMo
  dn=""
  cookie="<real_cookie>"
  inHierarchical="false">
  <inConfig>
    <aaaLdapEp
      attribute="CiscoAvPair"
      basedn="dc=pasadena,dc=company123,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"
  destSvc="mgmt-controller_dme"
  response="yes">
  <outConfig>
    <aaaLdapEp
      attribute="CiscoAvPair"
      basedn="dc=pasadena,dc=company123,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

The following 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: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>

```

Example

Note The descr property of orgDataCenter (under org-root/org-tenant1 and org-root/org-tenant2) 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.

The descr property of orgDataCenter (under org-root/org-tenant1 and org-root/org-tenant2) 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-tenant1" />
    <dn value="org-root/org-tenant2" />
  </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-tenant1/org-HR"
      fltAggr="0"
      level="2"
      name="HR"
      status="modified"/>
    </outConfigs>
  </configConfMoGroup>

```

configConfMos

The following 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>
```

Example

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
            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

The following 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>
```

Example

Request

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

Response

```
<configFindDnsByClassId
  cookie="<real_cookie>"
  commCookie="2/12/0/1810"
  srcExtSys="172.20.101.150"
  destExtSys="172.20.101.150"
  srcSvc="sam_extXMLApi"
  destSvc="service-reg_dme"
  response="yes"
  classId="eventRecord">
  <outDns>
    <dn value="event-log/10210"/>
    <dn value="event-log/10250"/>
    <dn value="event-log/10211"/>
    <dn value="event-log/10221"/>
    <dn value="event-log/10251"/>
    <dn value="event-log/10141"/>
    <dn value="event-log/10151"/>
  </outDns>
</configFindDnsByClassId>
```


configResolveChildren

The following 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>
```

Example

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

The following example shows how to return the requested managed object in a given class. If `inHierarchical = true`, the returned object will also contain its 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:complexType>
```

```

        </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>

```

Example

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

The following example shows how to return requested managed objects in several classes. If `inHierarchical= true`, the returned object will also contain its 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>
```

Example

Request

```
<configResolveClasses
  cookie="<real_cookie>"
  inHierarchical="false">
  <inIds>
    <Id value="eventRecord"/>
    <Id value="faultInst"/>
  </inIds>
</configResolveClasses>
```

Response

```
<configResolveClasses
  cookie="<real_cookie>"
  commCookie="2/12/0/181a"
  srcExtSys="172.20.101.150"
  destExtSys="172.20.101.150"
  srcSvc="sam_extXMLApi"
  destSvc="service-reg_dme"
```

```

response="yes">
<outConfigs>
  <eventRecord
    affected="observe/observed-1001-1"
    cause="transition"
    changeSet=""
    code="E4194388"
    created="2012-07-25T00:39:35.528"
    descr="[FSM:BEGIN]: Resolve Mgmt Controller
Fsm(FSM:sam:dme:ObserveObservedResolveControllerFsm) "
    dn="event-log/10133"
    id="10133"
    ind="state-transition"
    severity="info"
    trig="special"
    txId="3"
    user="internal"/>
  <eventRecord
    affected="observe/observed-1001-1"
    cause="transition"
    changeSet=""
    code="E4194388"
    created="2012-07-25T00:39:35.528"
    descr="[FSM:STAGE:END]:
(FSM-STAGE:sam:dme:ObserveObservedResolveControllerFsm:begin) "
    dn="event-log/10134"
    id="10134"
    ind="state-transition"
    severity="info"
    trig="special"
    txId="3"
    user="internal"/>
</outConfigs>
</configResolveClasses>

```

configResolveDn

The following 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>

```

```
</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>
```

Example

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="vpod-31.host123.com"
      intId="21959"
      name="vcenter1"
      operState="unknown"
      stateQual=""
      type="vmware"
      version=""/>
    </outConfig>
  </configResolveDn>
```

configResolveDns

The following example shows how to retrieve managed objects for a list of DNSs.

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>
```

Example

Request

```
<configResolveDns
  cookie="<real_cookie>"
  inHierarchical="false">
  <inDns>
    <dn value="sys" />
  </inDns>
</configResolveDns>
```

Response

```
<configResolveDns
  cookie="<real_cookie>"
  commCookie="5/12/0/1009"
```

```

srcExtSys="172.25.103.136"
destExtSys="172.25.103.136"
srcSvc="sam_extXMLApi"
destSvc="resource-mgr_dme"
response="yes">
<outConfigs>
  <topSystem
    address="172.25.103.136"
    currentTime="2012-08-01T00:47:44.663"
    dn="sys"
    mode="stand-alone"
    name="localhost"
    systemOrg=""
    systemUpTime="04:04:05:00"/>
</outConfigs>
<outUnresolved>
</outUnresolved>
</configResolveDns>

```

configResolveParent

The following 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: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>
```

Example

Request

```
<configResolveParent
  cookie="<real_cookie>"
  inHierarchical="false"
  dn="org-root/org-tenant1/org-HR">
</configResolveParent>
```

Response

```
<configResolveParent
  dn="org-root/org-tenant/org-HR"
  cookie="<real_cookie>"
  commCookie="2/12/0/1837"
  srcExtSys="172.20.101.150"
  destExtSys="172.20.101.150"
  srcSvc="sam_extXMLApi"
  destSvc="service-reg_dme"
  response="yes">
  <outConfig>
    <orgTenant
      descr="tenant123"
      dn="org-root/org-tenant"
      fltAggr="0"
      level="1"
      name="tenant123"/>
  </outConfig>
</configResolveParent>
```

configScope

The following 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>

```

Example

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

The following 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>

```

Example

Request

When the client application subscribes to an event or events using eventSubscribeApps or eventSubscribe, the VNMC sends 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

The following 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>

```

```
</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>
```

Example

Request

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

Response

VNMC sends no response or acknowledgement.

eventSubscribeApps

The following example shows a subscribe request for activity on specified applications. The client application can subscribe to the 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 VSG as well, where ip should be the IP address for the 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>
```

Example

Request

```
<eventSubscribeApps
```

```

cookie="<real_cookie>"
commCookie=""
srcExtSys="0.0.0.0"
destExtSys="0.0.0.0"
<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 the request is successful, VNMC sends no response or acknowledgement.

faultAckFault

The following 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>

```

Example

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

The following example shows how to send an acknowledgement 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>

```

Example

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

The following 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>
```

Example

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>
```

loggingSyncOcons

The following example shows how to retrieve event IDs from DME.

Request Syntax

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

Example

Request

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

Response

List of event IDs.

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, the returned objects will also contain their children. If inHierarchical = false, only the matching objects are returned. If inSingleLevel = true, only the objects at the starting organization level are returned. If inSingleLevel = false, objects in child organizations are also returned.

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:complexType>
```



```

<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>

```

Example

Request

```

<orgResolveElements
  dn="org-root/org-Cola"
  cookie="<real_cookie>"
  commCookie="7/15/0/19"

```

```
    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
      descr="The default Policy Set"
      dn="org-root/pset-default"
      intId="10082"
      name="default"/>
  </pair>
  <pair key="pset-myPolicySet3">
    <policyPolicySet
      descr=""
      dn="org-root/org-Cola/pset-myPolicySet3"
      intId="12289"
      name="myPolicySet3"/>
  </pair>
  <pair key="pset-policySetSanity">
    <policyPolicySet
      descr=""
      dn="org-root/org-Cola/pset-policySetSanity"
      intId="24627"
      name="policySetSanity"/>
  </pair>
  <pair key="pset-pci_compliance_f">
    <policyPolicySet
      descr=""
      dn="org-root/pset-pci_compliance_f"
      intId="24539"
      name="pci_compliance_f"/>
  </pair>
  <pair key="pset-pci_compliance_h">
    <policyPolicySet
      descr=""
      dn="org-root/pset-pci_compliance_h"
      intId="24541"
      name="pci_compliance_h"/>
  </pair>
</outConfigs>
</orgResolveElements>
```

orgResolveInScope

The following example shows how the system looks up the organization with the given DN, and (optional) parent organizations recursively to the root. If an organization is not found, 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: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: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:complexType>
```

```

    </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>

```

Example

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-vall"
            value="DEV"/>
          <policyAttributeDesignator
            attrName="dept"
            childAction="deleteNonPresent"

```

```

        rn="attr-ref"/>
      </policyVnspAVPair>
    </policyVirtualNetworkServiceProfile>
  </pair>
</outConfigs>
</orgResolveInScope>

```

poolResolveInScope

The following 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: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: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>

```

Example

Request

```

<poolResolveInScope
  dn="org-root/org-tenant1"
  cookie="<real_cookie>"
/>

```

Response

```

<poolResolveInScope
  dn="org-root/org-cisco"
  cookie="<real_cookie>"
  commCookie="5/12/0/19"
  srcExtSys="172.25.103.136"
  destExtSys="172.25.103.136"
  srcSvc="sam_extXMLApi"
  destSvc="resource-mgr_dme"
  response="yes">
  <outConfigs>
    <pair key="fwpool-default">
      <fwPool
        assigned="0"
        descr="Default Pool of firewall resources"
        dn="org-root/fwpool-default"
        fltAggr="65536"
        id="1"
        intId="10066"
        name="default"
        size="0"/>
    </pair>
  </outConfigs>
</poolResolveInScope>

```

Appendix: UML Diagrams

This section contains the following topics:

- [VPN Model](#)
- [Generic Rule-Based Policy Model](#)

VPN Model

Figure 4 VPN Model Granular Device View

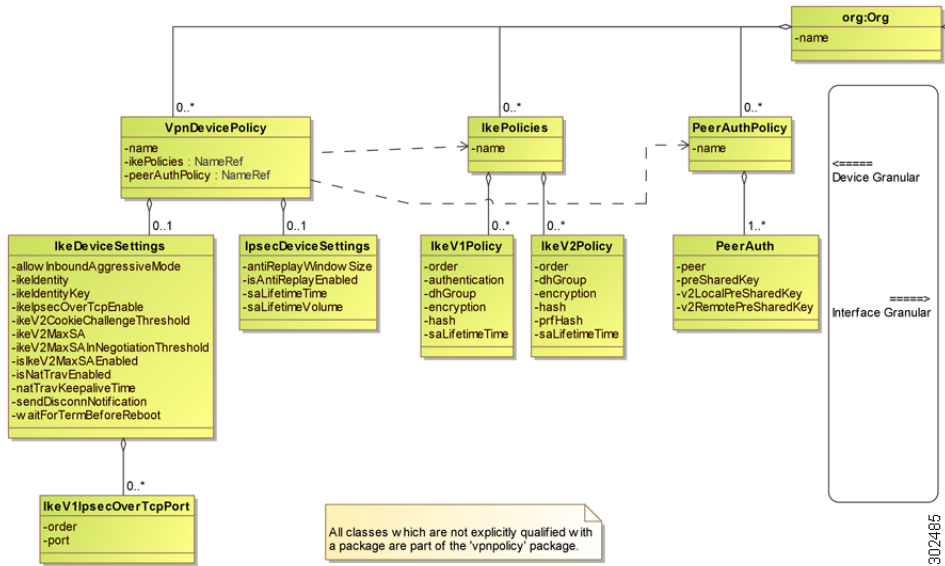
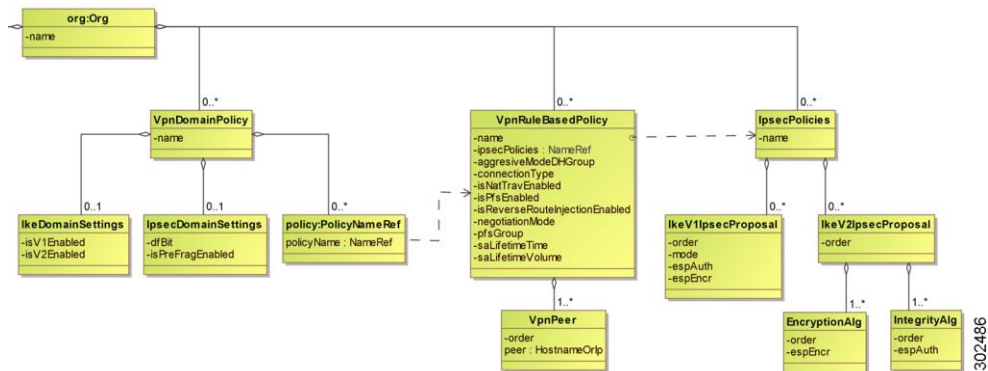
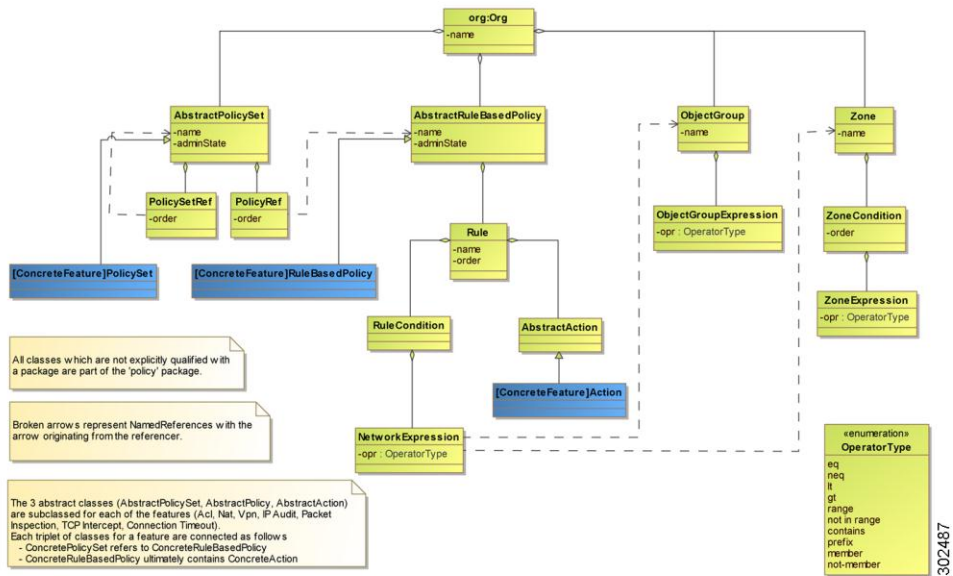


Figure 5 VPN Model Granular Interface View



Generic Rule-Based Policy Model

Figure 6 Generic Rule Based Policy Model



302487

Index

A

aaaGetRemoteUserRoles	49
aaaGetUserLocales	50
aaaKeepAlive	52
aaaLogin	52
aaaLogout	54
aaaRefresh	55
All Bits Filter	29
And Filter	30
And Or Not Composite Filters	31
Any Bits Filter	29
Attribute Dictionary	40
Authentication Request	32
Authentication Response	32

B

Between Filter	30
----------------------	----

C

Compute Security Profile	43
configCloneMo	57
configConfFiltered	58
configConfMo	13, 59
configConfMoGroup	13, 61
configConfMos	13, 62
configFindDnsByClassId	23, 65
configMoChangeEvent	14
configResolveChildren	23, 66
configResolveClass	23, 67
configResolveClasses	23, 68
configResolveDn	22, 70
configResolveDns	23, 72
configResolveParent	24, 73
configScope	24, 74
Create or Update Organization Request	33
Create or Update Organization Response	33

D

Device Profiles	36
-----------------------	----

E

Equality Filter	26
event notification	14
eventSendHeartbeat	76
eventSubscribe	76
eventSubscribeApps	77

F

False Conditions	26
faultAckFault	78
faultResolveFault	79

G

Greater Than Filter	27
Greater Than or Equal to Filter	27

L

Less Than Filter	27
Less Than or Equal to Filter	28
loggingSyncOcns	81
LogProfile Policy	36

M

Management Information Model	6
methodVessel	14
Modifier Filters	31

N

Not Equal Filter	27
Not Filter	31

O

Object Group	39
Or Filter	30
orgResolveElements	81
orgResolveInScope	84

P

Policy	40
PolicySet	42
poolResolveInScope	86

S

SNMP Policy	35
Syslog Policy	34

T

True Conditions	26
-----------------------	----

W

Wildcard Filter	28
-----------------------	----

Z

Zone	38
------------	----

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation* at: <http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html>.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation, as an RSS feed and deliver content directly to your desktop using a reader application. The RSS feeds are a free service.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Any Internet Protocol (IP) addresses used in this document are not intended to be actual addresses. Any examples, command display output, and figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses in illustrative content is unintentional and coincidental.

Cisco Virtual Network Management Center 2.1 XML API Reference Guide
OL-29821-01

© 2013 Cisco Systems, Inc. All rights reserved.