Cisco Security Manager 4.4 API Specification

(Version 1.1)

This document describes CSM message exchanges, XML schema, and client/server behavioral specifications in Cisco Security Manager 4.4 North Bound API

Version 1.0 Published: June 14, 2012
Version 1.0 Revised: July 10, 2012 (added sample programs to Section 8)
Version 1.1 Published: March 22, 2013 (added Unified and Trustsec support to Version 1.0)
## Table of Contents

1 Overview .................................................................................................................................11
   1.1 Scope ................................................................................................................................11
   1.2 Changes since previous version ........................................................................................12
      1.2.1 Unified Access Rules ..................................................................................................12
      1.2.2 Security Policy Object ..............................................................................................12
      1.2.3 Network object ...........................................................................................................12
      1.2.4 Return user/ticket that last modified a config rule .....................................................12
      1.2.5 Add device status – up/down as part of the event service ..........................................12
      1.2.6 Exec command API call will be supporting custom timeouts ......................................12
      1.2.7 API enhancement to return list of all the shared Policies defined in CSM ..................12
      1.2.8 Return the Device’s SysObjectID in the Device Object ............................................13
      1.2.9 CSM Audit Logs should differentiate between logins through API and CSM client .......13
      1.2.10 New Firewall Policies ...............................................................................................13
   1.3 Audience ............................................................................................................................13
   1.4 References ..........................................................................................................................13
   1.5 Glossary ..............................................................................................................................14
   1.6 Conventions .......................................................................................................................14
   1.7 Overview of CSM Message Flows ......................................................................................16
   1.8 Licensing ............................................................................................................................17
   1.9 Prerequisites .......................................................................................................................18
   1.10 API Administration Settings ............................................................................................18
   1.11 Debug Settings .................................................................................................................19
2 Common Service API ...............................................................................................................20
   2.1 Object Model ....................................................................................................................20
      2.1.1 Object Identifier .........................................................................................................20
      2.1.2 Base Object ...............................................................................................................20
      2.1.3 Device ........................................................................................................................21
      2.1.4 DeviceGroup ..............................................................................................................24
      2.1.5 Port Identifier .............................................................................................................25
      2.1.6 BaseError ...................................................................................................................26
   2.2 Methods .............................................................................................................................28
      2.2.1 Common Request & Response ..................................................................................28
      2.2.2 Method login .............................................................................................................29
      2.2.3 Method logout ...........................................................................................................34
      2.2.4 Method: ping .............................................................................................................35
3 CSM Configuration Service API ...............................................................................................38
   3.1 Object Model .....................................................................................................................38
      3.1.1 Base Policy ...............................................................................................................38
      3.1.2 BasePolicyObject .....................................................................................................41
      3.1.3 Policy Utility Classes ...............................................................................................43
      3.1.4 PolicyObject Derived Classes ..................................................................................45

Cisco Security Manager 4.4 API Specification (Version 1.1)
List of Figures

Figure 1: High Level CSM Message Exchange ........................................................................... 16
Figure 2: ObjectIdentifier & ObjectIdentifierList XML Schema .................................................. 20
Figure 3: BaseObject XML Schema ............................................................................................ 21
Figure 4: Device XML Schema ................................................................................................... 22
Figure 5: Interface XML Schema ................................................................................................ 24
Figure 6: DeviceGroup and DeviceGroupPath XML Schema ......................................................... 25
Figure 7: PortIdentifier XML Schema ......................................................................................... 26
Figure 8: BaseError XML Schema ............................................................................................... 26
Figure 9: BaseReqResp XML Schema .......................................................................................... 28
Figure 10: login Request Example ............................................................................................... 30
Figure 11: LoginRequest XML Schema ......................................................................................... 31
Figure 12: login Response Example ............................................................................................. 31
Figure 13: LoginResponse XML Schema Method heartbeatCallback ........................................... 32
Figure 14: Method heartbeatCallback Example ............................................................................ 33
Figure 15: logout Request Example ............................................................................................. 34
Figure 16: LogoutRequest XML Schema ....................................................................................... 35
Figure 17: logout Response Example .......................................................................................... 35
Figure 18: LogoutResponse XML Schema .................................................................................... 35
Figure 19: ping Request Example ................................................................................................ 36
Figure 20: PingRequest XML Schema ........................................................................................ 36
Figure 21: ping Response Example ............................................................................................. 37
Figure 22: PingResponse XML Schema ....................................................................................... 37
Figure 23: BasePolicy Class Inheritance ..................................................................................... 39
Figure 24: BasePolicy XML Schema ........................................................................................... 40
Figure 25: BasePolicyObject XML Schema ................................................................................ 42
Figure 26: Policy Utility Class XML Schema ............................................................................... 44
Figure 27: NetworkPolicyObject XML Schema ......................................................................... 46
Figure 28: IdentityUserGroupPolicyObject XML Schema ............................................................ 47
Figure 29: PortListPolicyObject XML Schema ........................................................................... 48
Figure 30: ServicePolicyObject XML Schema ............................................................................ 50
Figure 31: InterfaceRolePolicyObject XML Schema ................................................................. 51
Figure 32: TimeRangePolicyObject XML Schema ...................................................................... 53
Figure 99: Client Ping Processing Flowchart ................................................................. 155
Figure 100: Client Config Flowchart .............................................................................. 156
List of Tables

Table 1: Glossary Terms........................................................................................................14
Table 2: BaseObject Class Attributes ..............................................................................21
Table 3: Device Class Attributes ......................................................................................22
Table 4: Interface Class Attributes ...................................................................................23
Table 5: FirewallCapabilities Class Attributes .................................................................24
Table 6: DeviceGroup Class Attributes .............................................................................25
Table 7: PortIdentifier Class Attributes ..........................................................................25
Table 8: BaseError Class Attributes ................................................................................26
Table 9: System Error Codes ............................................................................................27
Table 10: BaseReqResp Class Attributes .........................................................................28
Table 11: login Request Elements and Attributes Descriptions ........................................30
Table 12: login Response Elements and Attributes Description ......................................32
Table 13: Method heartbeatCallback Elements and Attributes Description .......................33
Table 14: Login Method Error Codes ................................................................................34
Table 15: logout Request Elements and Attributes Descriptions .....................................34
Table 16: logout Response Elements and Attributes Description ....................................35
Table 17: ping Request Elements and Attributes Descriptions ........................................36
Table 18: ping Response Elements and Attributes Description ........................................37
Table 19: BasePolicy Class Attributes .............................................................................40
Table 20: BasePolicyObject Class Definition ....................................................................41
Table 21: NetworkPolicyObject Class Definition ...............................................................45
Table 22: IdentityUserGroupPolicyObject Class Definition .............................................46
Table 23: PortListPolicyObject Class Definition ...............................................................48
Table 24: ServicePolicyObject Class Definition ...............................................................49
Table 25: InterfaceRolePolicyObject Class Definition .....................................................51
Table 26: TimeRangePolicyObject Class Definition ........................................................52
Table 27: SLAMonitorPolicyObject Class Definition ........................................................54
Table 28: StandardACEPolicyObject Class Definition .....................................................56
Table 29: ExtendedACEPolicyObject Class Definition ...................................................57
Table 30: ACLPolicyObject Class Definition ...................................................................58
Table 31: SecurityGroupPolicyObject() Class Definition ..............................................59
Table 32: DeviceAccessRuleFirewallPolicy Class Definition .........................................62

Cisco Security Manager 4.4 API Specification (Version 1.1)
Table 33: DeviceAccessRuleUnifiedFirewallPolicy Class Definition ................................................................. 66
Table 34: DeviceStaticRoutingFirewallPolicy ................................................................................................. 67
Table 35: DeviceStaticRoutingRouterPolicy ................................................................................................. 69
Table 36: DeviceBGPRouterPolicy Class Definition ...................................................................................... 72
Table 37: InterfaceNATRouterPolicy Class Definition .................................................................................... 73
Table 38: InterfaceNATStaticRulesRouterPolicy Class Definition ............................................................. 75
Table 39: InterfaceNATDynamicRulesRouterPolicy Class Definition ......................................................... 77
Table 40: DeviceNATTimeoutsRouterPolicy Class Definition ....................................................................... 79
Table 41: InterfaceNATAddressPoolFirewallPolicy Class Definition .......................................................... 81
Table 42: DeviceNATTransOptionsFirewallPolicy Class Definition ............................................................ 82
Table 43: InterfaceNATTransExemptionsFirewallPolicy Class Definition .................................................. 84
Table 44: InterfaceNATDynamicRulesFirewallPolicy Class Definition ...................................................... 85
Table 45: InterfaceNATPolicyDynamicRulesFirewallPolicy Class Definition ............................................. 88
Table 46: InterfaceNATStaticRulesFirewallPolicy Class Definition ........................................................... 91
Table 47: InterfaceNATManualFirewallPolicy Class Definition .................................................................. 96
Table 48: InterfaceNAT64ManualFirewallPolicy Class Definition .............................................................. 98
Table 49: InterfaceNATObjectFirewallPolicy Class Definition .................................................................... 100
Table 50: InterfaceNAT64ObjectFirewallPolicy Class Definition .................................................................. 102
Table 51: Method GetServiceInfo Request URL Argument Descriptions .................................................. 104
Table 52: GetServiceInfo Response Elements and Attributes Description .................................................. 105
Table 53: Method GetGroupList Request URL Argument Descriptions ....................................................... 106
Table 54: GetGroupList Response Elements and Attributes Description ..................................................... 108
Table 55: GetGroupList Method Error Codes ............................................................................................... 109
Table 56: Method GetDeviceListByCapability Request URL Argument Descriptions .............................. 110
Table 57: GetDeviceListByCapability Response Elements and Attributes Description ............................... 111
Table 58: Method GetDeviceListByGroup Request URL Argument Descriptions ...................................... 113
Table 59: Method GetDeviceConfigByGID Request URL Attribute Descriptions ........................................ 115
Table 60: GetDeviceConfigByGID Response Elements and Attributes Description ..................................... 117
Table 61: GetDeviceConfigByGID Method Error Codes ............................................................................... 118
Table 62: Method GetDeviceConfigByName Request URL Attribute Descriptions .................................... 119
Table 63: GetDeviceConfigByName Method Error Codes .......................................................................... 120
Table 64: Method GetPolicyListByDeviceGID Request URL Argument Descriptions ............................... 122
Table 65: GetPolicyListByDeviceGID Response Elements and Attributes Description ............................. 123
Table 66: GetPolicyListByDeviceGID Method Error Codes ........................................................................ 124
Table 67: Method GetPolicyConfigByName Request URL Attribute Descriptions .............................................. 125
Table 68: GetPolicyConfigByName Response Elements and Attributes Description ..................................... 128
Table 69: GetPolicyConfigByName Method Error Codes ............................................................................... 128
Table 70: Method GetPolicyConfigByDeviceGID Request URL Attribute Descriptions ......................... 129
Table 71: Method getSharedPolicyNamesByType Request URL Attribute Descriptions ......................... 131
Table 72: GetSharedPolicyNamesByType Response Elements and Attributes Description ..................... 133
Table 73: eventSubscription Request Elements and Attributes Descriptions ........................................ 135
Table 74: eventSubscription Response Elements and Attributes Description ........................................ 138
Table 75: EventSubscription Method Error Codes ..................................................................................... 139
Table 76: ConfigChangeEvent Data Element Descriptions .......................................................................... 140
Table 77: DeviceStatusEvent Data Element Descriptions ......................................................................... 143
Table 78: Method execDeviceReadOnlyCLICmds Request Elements and Attributes Descriptions ............. 149
Table 79: execDeviceReadOnlyCLICmds Response Elements and Attributes Description ....................... 151
Table 80: ExecDeviceReadOnlyCLICmdsRequest Method Error Codes ..................................................... 152
1 Overview

This document provides the protocol description and specification for the Cisco Security Manager (CSM) API as well as the behavioral requirements for any CSM client products that will use the API on the CSM Server and CSM Server (infrastructure device) itself. In addition, it provides the XML schema, which is the basis for the message content carried by the CSM northbound (NB) API.

The CSM NB API is designed to be used by client products that may wish to read network security configuration information or events as well in the future of publishing changes to those configurations.

CSM NB API is broken into services that enable various features providing access to both global as well as device specific network configuration policies, for Cisco ASA and IPS devices.

After a client receives the policy information, it can use the information to perform security analysis, link to security events…etc.

1.1 Scope

This document serves as the CSM NB API 1.1 specification, the specification for the XML schema which provides the basis of the message content carried in CSM API payload and the behavioral specification for CSM Client products and infrastructure products implementing the CSM NB API.

Note: Unless otherwise stated, all references to CSM Client or CSM "API" Client throughout this document refer to the third party "CSM API Client Program" that uses the REST interface to communicate with the CSM Server. The CSM client mentioned in this document should not be confused with the pre-bundled CSM GUI client application that is installed on user desktops to access the server. Wherever necessary, any references to the pre-bundled CSM GUI client application in the document includes a screen shot of the relevant screen to prevent any confusion. Please see the figure below where the interaction of the CSM API Client Program is clearly denoted.
1.2 Changes since previous version

The NB API 1.1 is a successor of NB API 1.0. The following changes have been added in the 1.1 version of the API.

1.2.1 Unified Access Rules
This is a new policy, supported in this version.

1.2.2 Security Policy Object
This is a new policy object, supported in this version.

1.2.3 Network object
This object has been changed to represent its data elements as <ipData>, in place of <ipv4Data>, when API is used for Unified Access Rules.

1.2.4 Return user/ticket that last modified a config rule.
One of the use cases of API is to get configuration data from CSM for compliance checks. The configuration service API now also returns information regarding users and tickets that modified a configuration rule.

1.2.5 Add device status – up/down as part of the event service
Some Security Manager users have task threads that run daily and fetch changed configuration from the network devices managed by Security Manager. For this they use the execDeviceReadOnlyCLICmds API call to get the configuration from the Device.

The execDeviceReadOnlyCLICmds API call hangs if the Device Status is down. Hence, users interested in this area would like to do a Device Status up/down check before executing the execDeviceReadOnlyCLICmds API call.

1.2.6 Exec command API call will be supporting custom timeouts.
In the previous version of the API (version 1.0), execDeviceReadOnlyCliCmds has no timeout value, so it can run for an infinite period of time and cause the API service to hang if a device is unresponsive. To prevent that problem, the current version of the API (version 1.1) allows the method execDeviceReadOnlyCliCmds to take an optional attribute in the Request. This optional attribute allows the API client to set an timeout for the execDeviceReadOnlyCliCmds method call.

1.2.7 API enhancement to return list of all the shared Policies defined in CSM.
A new API has now been added in version 1.1 that returns the list of all shared policies in the system for a given policy type.
1.2.8 Return the Device’s SysObjectID in the Device Object.
API version 1.0 has many APIs which list out all the devices present in CSM. Additionally, API version 1.1 now includes the SysObjectID in the response.

1.2.9 CSM Audit Logs should differentiate between logins through API and CSM Client.
In CSM 4.3 Logins to CSM from NB API or a CSM Client would create an Audit Log message indicating a user Login. However there was no differentiation/description in the Audit Log message that a user has logged in using NB API.
As part of the CSM 4.4 release, the audit log is now enhanced to differentiate between logins through API and CSM Client.

1.2.10 New Firewall Policies
The following new firewall policies are available beginning with CSM 4.4 and version 1.1 of the API:

- InterfaceNAT64ObjectFirewallPolicy. An InterfaceNAT64ObjectFirewallPolicy represents a Unified (IPv6/IPv4) Object NAT Rule.

1.3 Audience
This document is technical in nature and is intended for architects, development engineers and technical marketing engineers.

1.4 References
The following are referenced within this document and form a normative part of this specification to the extent specified herein. In the event of a conflict with this specification and the following referenced specifications, the contents of this specification take precedence.

- RFC-3579, RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP), Aboba and Calhoun, September 2003
- Describing Media Content of Binary Data in XML, Karmarkar and Yaçınalp, http://www.w3.org/TR/xml-media-types/, May 4, 2005
- RFC-2616, Hypertext Transfer Protocol -- HTTP/1.1, Fielding, Gettys, Mogul, Frystyk, Masinter, Leach and Berners-Lee, June 1999
- XML Path Language http://www.w3.org/TR/xpath/, November 1999

Cisco Security Manager 4.4 API Specification (Version 1.1)
1.5 Glossary

The following list describes acronyms and definitions for terms used throughout this document:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>authentication, authorization and accounting</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AS</td>
<td>authentication server (e.g. AAA)</td>
</tr>
<tr>
<td>AUS</td>
<td>Auto Update Server</td>
</tr>
<tr>
<td>CA</td>
<td>certificate authority. May also refer to &quot;Configuration Archive&quot; module in CSM that stores raw device configuration data.</td>
</tr>
<tr>
<td>CNS</td>
<td>Cisco Networking Services</td>
</tr>
<tr>
<td>CR</td>
<td>certificate repository</td>
</tr>
<tr>
<td>CSM</td>
<td>Cisco Security Manager</td>
</tr>
<tr>
<td>DNS</td>
<td>domain name service</td>
</tr>
<tr>
<td>FQDN</td>
<td>fully qualified domain name</td>
</tr>
<tr>
<td>HPM</td>
<td>Health and Performance Monitoring application of CSM</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet protocol</td>
</tr>
<tr>
<td>LAN</td>
<td>local area network</td>
</tr>
<tr>
<td>NAS</td>
<td>network authentication server</td>
</tr>
<tr>
<td>NB</td>
<td>North Bound</td>
</tr>
<tr>
<td>OOB</td>
<td>Out of Band (uncontrolled changes done outside of CSM directly on the device)</td>
</tr>
<tr>
<td>PKI</td>
<td>public key infrastructure</td>
</tr>
<tr>
<td>PKC</td>
<td>public key cryptography</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>UI</td>
<td>user interface</td>
</tr>
<tr>
<td>URI</td>
<td>universal resource identifier</td>
</tr>
<tr>
<td>XML</td>
<td>extensible markup language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
</tbody>
</table>

Table 1: Glossary Terms

1.6 Conventions

The following textual conventions are following in this document.

- All type definitions start with an uppercase letter
  - E.g. xs:simpleType name="ObjectIdentifier"
- All element names start with a lowercase letter
  - E.g `<xs:element name="gid" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>`
1.7 Overview of CSM Message Flows

When a CSM client wishes to use the CSM server, some of the main high-level message flows are shown in Figure 1. The CSM Server is configured to know about the devices in the network and reads those configurations for Firewalls, IPS…etc. After initiation any CSM client may authenticate to the CSM server and access the methods provided in the API. The CSM interface is defined as multiple services such as common (section 2), config (section 3), events (section 4) and utility (section 5) services.

**Figure 1: High Level CSM Message Exchange**

REST is used for the transport of CSM messages. In other words, a CSM message is a URL accessed by the CSM client. The XML schema for CSM messages is provided in each service section in this document as well as the complete XSD files in section 10.
CSM API is structured across four major sub-categories:

- **Common Service API**: Contains common method definitions and schema details (related to login, logout, ping/heartbeat etc.). For more details see section titled Common Service API
- **Config Service API**: Contains methods and schema to retrieve device configuration data (read-only). For more details see section titled CSM Configuration Service API
- **Event Service API**: Contains methods and schema that allow the client to subscribe for event notifications in specific cases (deployment and out of band change events etc.). For more details see section titled CSM Events Service API
- **Utility Service API**: Contains methods and schema that allow the user to execute specific CLI commands on the device using the CSM API interface. For more details see section titled CSM Utility Service API

### 1.8 Licensing

This feature is licensed. A specific CSM API license must be applied via the CSM **Tools → Security Manager Administration → Licensing** page. The API license can only be applied on a server licensed for the CSM professional edition. The license cannot be applied on CSM running a Standard edition of the license or when it is in evaluation mode.
All API requests without a valid license will return an error response containing:

**Error code:** 26

**Error Description:** API license is not enabled. Please add a valid API license and retry this operation.

### 1.9 Prerequisites

Following are some prerequisites to use and work with the API:

- Latest version of the CSM Server that includes the API support must be installed. Please refer to the CSM Install Guide for more details.
- After the Server is installed, managed devices (ASA/IPS) must be added to CSM before the corresponding policy data is queried using the API.
- API will only return data that is “committed” to the policy database. So applicable activities must be submitted before the data is visible via the API.
- Health and Performance Monitoring (HPM) module should be enabled for Out of Band (OOB) and device status UP/DOWN change notifications to work.
- API does not include any SDK for any specific programming language. The API exposes a REST (Representational State Transfer) based interface, so API clients can be implemented in any language as long as the XML Message protocol is adhered to.
- All API Services are accessible only using HTTPS. No HTTP access is allowed. The “root” URL for REST based API access is `https://<server-ip or host>/nbi/`.

### 1.10 API Administration Settings

Following Global administration settings are specific to the API feature (Please refer to the version specific CSM user and/or install guide for additional information):
- **Enable API Service**: Allows the API feature to be completely enabled/disabled
- **Result Set Page Size**: Controls the size of the result set that is returned in a single response. Values are 100 (minimum), 500 (default) and 1000 (maximum). This configuration only applies to the `GetPolicyConfigurationByName` (section 3.2.8) and `GetPolicyConfigurationByDeviceGID` (section 3.2.9) methods. Also see section 2.2.1.1 for details on how responses are paginated.
- **Active Client Sessions**: Controls the total number of simultaneous active API client sessions (logins) that are allowed. Values are 1 (minimum), 5 (default) and 10 (maximum).

### 1.11 Debug Settings

If you want to enable debugging on the API requests on the CSM server, please set the value of the API Debug Level field to DEBUG. (The default value of this field is ERROR.)
2 Common Service API

This section describes the common methods and common object model for all services on the CSM API.

2.1 Object Model

The following object classes are used throughout the API specification.

2.1.1 Object Identifier

The Object Identifier is a global unique identifier for an object. The Object Identifier is a 128-bit value based on RFC4122.

```
<xs:simpleType name="ObjectIdentifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="ObjectIdentifierList">
  <xs:sequence>
    <xs:element name="gid" type="ObjectIdentifier" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Figure 2: ObjectIdentifier & ObjectIdentifierList XML Schema

2.1.2 Base Object

The common class for all objects in the system and all object classes inherit from this class. It has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>Object Identifier</td>
<td>Unique object identifier for an object and is immutable for the object lifetime</td>
</tr>
<tr>
<td>lastUpdateTime</td>
<td>TimeStamp</td>
<td>Indicates the update time for an object either created or updated.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>An optional display name for the object.</td>
</tr>
<tr>
<td>parentGID</td>
<td>Object Identifier</td>
<td>An optional parent object identifier that identifies an object’s parent instance.</td>
</tr>
<tr>
<td>updatedByUser</td>
<td>String</td>
<td>Username of the User who updated the Object.</td>
</tr>
<tr>
<td>lastCommitTime</td>
<td>dateTime</td>
<td>Last Updated time for the object</td>
</tr>
<tr>
<td>ticketId</td>
<td>String</td>
<td>Ticket ID of the ticket as part of which the Object has been updated.</td>
</tr>
</tbody>
</table>
This will not be available if Ticketing is not enabled.

| activityName | String | Activity Name of the Activity as part of which the Object has been updated. This is applicable if the CSM is in Workflow mode. |

Table 2: BaseObject Class Attributes

```xml
<x:s:complexType name="BaseObject">
  <xs:s:sequence>
    <xs:s:element name="gid" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
    <xs:s:element name="name" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:s:element name="lastUpdateTime" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
    <xs:s:element name="parentGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
    <xs:s:element name="updatedByUser" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:s:element name="lastCommitTime" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
    <xs:s:element name="ticketId" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:s:element name="activityName" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:s:sequence>
</xs:s:complexType>
```

Figure 3: BaseObject XML Schema

2.1.3 Device

A device object is used to represent a single device in the system. Virtual contexts are also modeled as individual devices in the system. The device object also expresses a relationship between a parent (admin/system context) and its associated child contexts.

The Device class inherits from BaseObject including all attributes.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>osType</td>
<td>String</td>
<td>The device OS Type from an enumerated list {IOS, FWSM, ASA, PIX, IPS}</td>
</tr>
<tr>
<td>osVersion</td>
<td>String</td>
<td>The software version of the OS running on the device. Version string could be 6.1, 6.2 etc. on a PIX platform, 12.1, 12.2S etc. on an IOS platform etc.</td>
</tr>
<tr>
<td>imageName</td>
<td>String</td>
<td>The OS Image name.</td>
</tr>
<tr>
<td>mgmtInterface</td>
<td>Interface</td>
<td>A reference to the management interface for the device. The interface which is used to manage the device in CSM is taken as management interface.</td>
</tr>
<tr>
<td>interfaceList</td>
<td>Sequence of Interface</td>
<td>A list of interfaces within the device. (Contains list of interfaces other than management interface of the device.)</td>
</tr>
<tr>
<td>fullConfig</td>
<td>String</td>
<td>An element containing the full configuration of the device. The</td>
</tr>
</tbody>
</table>
full configuration of the device will be represented by an ASCII output of the show running config CLI command. This is only shown when the `getDeviceConfigByGid` or `getDeviceConfigByName` API’s to get the full config of the device. In all other cases this element value is not set.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualContextList</td>
<td>List of Device A list of virtual context objects belonging to the device</td>
</tr>
<tr>
<td>configState</td>
<td>ConfigurationSTate  The configuration state of the device and whether there are any uncommitted changes. Value taken from { committed, deployed }</td>
</tr>
<tr>
<td>sysObjectID</td>
<td>String System Object ID of the Device.</td>
</tr>
</tbody>
</table>

Table 3: Device Class Attributes

```
<xs:complexType name="Device">
  <xs:complexContent>
    <xs:extension base="BaseObject">
      <xs:sequence>
        <xs:element name="osType" type="OSType" minOccurs="1" maxOccurs="1"/>
        <xs:element name="osVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="imageName" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="sysObjectID" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="fullConfig" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="mgmtInterface" type="Interface" minOccurs="0" maxOccurs="1"/>
        <xs:element name="interfaceList" type="InterfaceList" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="virtualContextList" type="Device" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="configState" type="ConfigurationState" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 4: Device XML Schema

Cisco Security Manager 4.4 API Specification (Version 1.1)
### 2.1.3.1 Interface

The Interface class defines a network interface in a network device. The InterfaceList class defines a sequence of Interface instances.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>The type of network interface. Following are the supported interface type values: Null, Management, Analysis-module, Async, ATM, BRI, BVI, Content-engine, Dialer, Dot11Radio, Ethernet, FastEthernet, GigabitEthernet, TenGigabitEthernet, HundredGigabitEthernet, FDDI, Group-Async, HSSI, IDS-Sensor, Loopback, Multilink, Port-channel, POS, PRI, Serial, Switch, Tokenring, Tunnel, VG-anylan, Virtual-Template, Virtual-TokenRing, VLAN, Redundant</td>
</tr>
<tr>
<td>identifier</td>
<td>String</td>
<td>The identifier for the network interface. If name is configured for any interface on the device then Interface Identifier value will be taken as name else the interface type/port/slot will be shown as identifier. For example a GigabitEthernet interface is configured as outside then the API will show the identifier as outside and type as GigabitEthernet.</td>
</tr>
<tr>
<td>ipInterface.domainName</td>
<td>String</td>
<td>An optional DNS domain name.</td>
</tr>
<tr>
<td>ipInterface.ipAddress</td>
<td>String</td>
<td>The IP Address configured on the interface including its mask. The mask is not stored for IPS devices.</td>
</tr>
<tr>
<td>ipInterface.isNatAddress</td>
<td>String</td>
<td>True if the management IP address is a NATed address. (This placeholder element is not used in version 1.1 of the API)</td>
</tr>
<tr>
<td>ipInterface.realIpAddress</td>
<td>String</td>
<td>In case the management IP address is a NATed address, the realIpAddress is the non NATed ip address of the management interface. (This placeholder element is not used in version 1.1 of the API)</td>
</tr>
<tr>
<td>macInterface.macAddress</td>
<td>String</td>
<td>The mac address of the interface (optional). The MAC Address for ASA/PIX devices alone is stored. For others device types, CSM does not store the MAC address. Also CSM does not manage the burnt in MAC Address details. CSM manages only the mac - address that is changed explicitly by the user, so response will have only this data.</td>
</tr>
</tbody>
</table>

**Table 4: Interface Class Attributes**
Figure 5: Interface XML Schema

2.1.3.2 Firewall Capabilities

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>fwOsMode</td>
<td>Int</td>
<td>Indicates if the FWSM/ASA is running in Transparent (1) or Routed (2) mode.</td>
</tr>
<tr>
<td>fwOsMultiplicity</td>
<td>Int</td>
<td>Returns if the FWSM/ASA is running in a single (1) or multi (2) context.</td>
</tr>
<tr>
<td>contextName</td>
<td>String</td>
<td>Returns the name of the context. If the current device represents a context.</td>
</tr>
<tr>
<td>isComposite</td>
<td>Boolean</td>
<td>Indicates if the device is a “composite” or not. Catalyst 6500 containing FWSM or other blades are marked as composite.</td>
</tr>
</tbody>
</table>

Table 5: FirewallCapabilities Class Attributes

2.1.4 DeviceGroup

A device group object is used to represent a container of devices in the system. Device groups contain zero or more devices and zero or more children device groups.
The DeviceGroup class inherits from BaseObject including all attributes.

<table>
<thead>
<tr>
<th>Element/Attributes</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element: path</td>
<td>String</td>
<td>A hierarchical path string indicating the group.</td>
</tr>
<tr>
<td>Element: device</td>
<td>Device</td>
<td>Zero or more devices within this device group</td>
</tr>
<tr>
<td>Element: deviceGroup</td>
<td>DeviceGroup</td>
<td>Zero or more device groups within this device group</td>
</tr>
</tbody>
</table>

Table 6: DeviceGroup Class Attributes

```xml
<xs:complexType name="DeviceGroup">
  <xs:complexContent>
    <xs:extension base="BaseObject">
      <xs:sequence>
        <xs:element name="path" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="device" type="Device" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="deviceGroup" type="DeviceGroup" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 6: DeviceGroup and DeviceGroupPath XML Schema

### 2.1.5 Port Identifier

The PortIdentifier class defines a physical or virtual port identifier in a network device.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>slotNum</td>
<td>unsignedInt</td>
<td>The identifier of the slot in the case of a modular chassis. In a non-modular chassis this attribute is empty.</td>
</tr>
<tr>
<td>moduleNum</td>
<td>unsignedInt</td>
<td>The identifier of the module in the case of a sub-module within a slot. In a non-modular chassis this attribute is empty.</td>
</tr>
<tr>
<td>portNum</td>
<td>unsignedInt</td>
<td>The port number</td>
</tr>
</tbody>
</table>

Table 7: PortIdentifier Class Attributes
2.1.6 BaseError

The common class for all request errors is defined by the BaseError class. It has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Unsigned Long</td>
<td>A unique error code that identifies the type of error that occurred with the request.</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>A description of the error that occurred</td>
</tr>
</tbody>
</table>

The following general error codes are currently defined:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>General Failure</td>
</tr>
<tr>
<td>2</td>
<td>Lack of resources</td>
</tr>
<tr>
<td>3</td>
<td>Object Creation Failure</td>
</tr>
<tr>
<td>4</td>
<td>Authorization Failure: No session found</td>
</tr>
<tr>
<td>5</td>
<td>Authorization Failure: Invalid or expired session</td>
</tr>
<tr>
<td>6</td>
<td>Internal communication failure</td>
</tr>
<tr>
<td>13</td>
<td>XML request payload contains no data.</td>
</tr>
</tbody>
</table>

Figure 7: PortIdentifier XML Schema

Figure 8: BaseError XML Schema
XML request is invalid. (This error is caused if the XML request does not adhere to the published XML schema or if the XML is illformed or invalid. Though this is a system wide error, this error is usually set inside a specific response object if the application is able to parse the request method that is being called)

protVersion is optional and if an unsupported version is specified the this error is returned. Supported version is 1.0 in this release.

Internal Error

API Service is disabled

API license is not applied

<table>
<thead>
<tr>
<th>Table 9: System Error Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional method specific errors codes are defined in the respective sections. All response objects extend from the Base Error object. The error content is set in cases the CSM API encounters any error when servicing a method.</td>
</tr>
<tr>
<td>The error messages returned are of two general types:</td>
</tr>
<tr>
<td>1. <strong>Common/System wide errors</strong>: As defined in the table above. These errors are generally due to some unrecoverable errors encountered by the application. These errors can occur when servicing any request.</td>
</tr>
<tr>
<td>2. <strong>Method Specific errors</strong>: These are application errors that are specific to method being processed.</td>
</tr>
<tr>
<td>The system wide errors return the Base Error content as the only response:</td>
</tr>
</tbody>
</table>

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:baseError xmlns:ns1="csm">
  <code>1</code>
  <description>General Failure</description>
</ns1:baseError>
```

Method specific errors encode the Base Error content inside the response object. Following is an example of an error encountered when servicing a login request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:loginResponse xmlns:ns1="csm">
  <protVersion>1.0</protVersion>
  <error>
    <code>7</code>
    <description>Authentication Failure: Invalid username and/or password specified.</description>
  </error>
  <serviceVersion>1.0</serviceVersion>
  <sessionTimeoutInMins>15</sessionTimeoutInMins>
</ns1:loginResponse>
```
2.2 Methods

2.2.1 Common Request & Response

All methods in this API take an XML object as an argument for the method request and response. The XML object passed as part of the request (and echoed in the response) are derived from the following class which includes the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>protVersion</td>
<td>double</td>
<td>identifies the version of the protocol associated with the particular request/response being sent</td>
</tr>
<tr>
<td>reqId</td>
<td>String</td>
<td>identifies a unique token sent by the client in the request that the server will echo in the associated response</td>
</tr>
<tr>
<td>startIndex</td>
<td>Unsigned Long</td>
<td>An optional start index that is specified by the &quot;client&quot; requests that are &quot;paginated&quot;. This is applicable for policies like firewall rules that might return a large number of rows of data. The client must set the startIndex equal to the end index of the previous response to fetch the next page of data.</td>
</tr>
<tr>
<td>endIndex</td>
<td>Unsigned Long</td>
<td>An optional end index that is specified by the &quot;server&quot; when it has not returned “all the data”.</td>
</tr>
<tr>
<td>totalCount</td>
<td>Unsigned Long</td>
<td>An optional total count from the “server” that indicates the total number of rows in this policy.</td>
</tr>
<tr>
<td>Error</td>
<td>BaseError</td>
<td>identifies any errors that may occur upon a request submitted to the server</td>
</tr>
</tbody>
</table>

Table 10: BaseReqResp Class Attributes

```xml
<xs:complexType name="BaseReqResp">
   <xs:sequence>
      <xs:element name="protVersion" type="xs:double" minOccurs="0" maxOccurs="1"/>
      <xs:element name="reqId" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="startIndex" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
      <xs:element name="endIndex" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
      <xs:element name="totalCount" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
      <xs:element name="error" type="BaseError" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:complexType>
```

Figure 9: BaseReqResp XML Schema

2.2.1.1 Pagination
Some of the service calls (say fetching a list of access rules) might return a large set of data. To prevent such large fetches from causing performance issues on the server and client, these results will be ‘paged’. Pagination is applicable only for the `GetPolicyConfigByName` and `GetPolicyConfigByDeviceGID` methods. The pagination scheme will work as follows:

1) In the first request the client sends, none of the parameters `startIndex`, `endIndex` will be set. This is an indication to the server that this is a fresh request.

2) If the server determines that this request needs to be paginated (because the total result set is greater than the page size) then it will return a paged result and will set the following two elements:
   a. **endIndex**: Will be set to endIndex of the current result set. For example if this is the first request and 1000 rows are being returned then endIndex will be 1000.
   b. **totalCount**: Will contain the total count of the query itself. For example, if the total result contains 10,000 rows, then the element `totalCount` will be set to 10,000.

3) In the subsequent request, the client must set the `startIndex` in the BaseReqResp object and send the same query request again. In this case the `startIndex` of the client request will be equal to the `endIndex` of the last response received.

4) In the final response if there are no more data to be fetched, the server will not set any data for `endIndex` and `totalCount`.

Consider an example where a client is querying Firewall rules on a device “A” which has 3600 rules. The page size configured in the system is 1000. Following are the sequence of calls:

- **Client** → Get Firewall Rules on device “A”
- **Server** → Response with 1000 rules and `endIndex`=1000 and `totalCount`=3600
- **Client** → Get Firewall Rule on device “A” and `startIndex`=1000
- **Server** → Response with 1000 rules and `endIndex`=2000 and `totalCount`=3600
- **Client** → Get Firewall Rule on device “A” and `startIndex`=2000
- **Server** → Response with 1000 rules and `endIndex`=3000 and `totalCount`=3600
- **Client** → Get Firewall Rule on device “A” and `startIndex`=3000
- **Server** → Response with 600 rules and `endIndex`=<not-set> and `totalCount`=<not-set>

As a general case, if the client notices that the `endIndex` or `totalCount` is not set then all rows are expected to have been returned.

### 2.2.2 Method login

The login method authenticates a CSM client attempting to access the services provided by the CSM server. This method must be called prior to any other method called on other services.

#### 2.2.2.1 Request

An example of the method login request is shown in the figure below. The fields in these messages are described in the table below.
Figure 10: login Request Example

Table 11: login Request Elements and Attributes Descriptions

<table>
<thead>
<tr>
<th>XML Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginRequest</td>
<td>login request authenticates the client against the server and returns a cookie that should be used in subsequent requests</td>
</tr>
<tr>
<td>username</td>
<td>The CSM client username associated with the session</td>
</tr>
<tr>
<td>password</td>
<td>The CSM client password associated with the session</td>
</tr>
<tr>
<td>heartbeatRequested</td>
<td>This attribute may be optionally defined. If the attribute is set to true then the CSM client will receive a heartbeat callback from the CSM server. The server will try to ping the client with a frequency close to (\text{inactivity timeout} / 2) minutes. If the client does not respond to the heartbeat then the API retries the heartbeat during the next interval. If the heartbeat is successful then the session inactivity timeout is reset.</td>
</tr>
<tr>
<td>callbackUrl</td>
<td>The URL at which the CSM server will make the callback. This needs to be specified if the heartbeatRequested is true. Only HTTPS based callback URLs are allowed</td>
</tr>
</tbody>
</table>

HTTP Method  POST

Returns  200 OK + XML

401 Unauthorized
2.2.2.2 Response

The Login API validates the user credentials and returns a session token as a secure cookie. The session value is stored under the “asCookie” key. This session has a default session inactivity timeout of 15 minutes. An example of the HTTP Header and XML content response is shown in figure below. The fields in these messages are described in table below.

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Content:
<?xml version="1.0" encoding="UTF-8"?>
<loginResponse>
  <protVersion>1.0</protVersion>
  <serviceVersion>1.0.1</serviceVersion>
  <sessionTimeoutInMins>15</sessionTimeoutInMins>
</loginResponse>

serviceVersion will be 1.1 for the following Services for:
- Config Service
- Events Service
- Utility Service

Figure 12: login Response Example
Table 12: login Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>HTTP/XML Response</th>
<th>Definition</th>
</tr>
</thead>
</table>
| HTTP Header: asCookie | A cookie defining the authentication string, expiration date, path and domain that must be passed in all subsequent method calls  
  e.g. Set-Cookie: asCookie=732423dfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com |
| XML Element: loginResponse | XML content that returns the session information or failure |
| serviceVersion       | The service version of the Configuration service running. This attribute is only included if the user is successfully authenticated. |
| sessionTimeoutInMins | The session timeout in minutes is the amount of minutes that must pass with no activity by the client before the CSM server discards the session as no longer active. Any access by the CSM client after a session has been timed out will be rejected and require the client to re-authenticate to gain access to the CSM server again. This attribute is only included if the user is successfully authenticated. Default is 15 minutes. |

```xml
<xs:element name="loginResponse" type="LoginResponse"/>
<xs:complexType name="LoginResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="serviceVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="sessionTimeoutInMins" type="xs:positiveInteger" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 13: LoginResponse XML Schema Method heartbeatCallback

An example of the Method heartbeatCallback is shown below. The fields in these messages are described in the table below.
**Figure 14: Method heartbeatCallback Example**

**Table 13: Method heartbeatCallback Elements and Attributes Description**

<table>
<thead>
<tr>
<th>HTTP Header/XML Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>heartbeatCallbackRequest</td>
<td>The heartbeatCallback method is called by the CSM Server on the CSM client to ensure the csm client is still active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>POST</th>
</tr>
</thead>
</table>

| HTTP Header: asCookie     | The cookie returned by the login method that identifies the authentication session |

<table>
<thead>
<tr>
<th>Returns</th>
<th>200 OK</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>401 Unauthorized</th>
</tr>
</thead>
</table>

**Method specific errors:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Authentication Failure: Invalid username and/or password specified</td>
</tr>
<tr>
<td>8</td>
<td>Authorization Failure: Login session error. The asCookie contained in the request is incorrect. This error can also occur if the session is stale (invalidated/timed out). The session can be kept alive by using ping or heartbeat.</td>
</tr>
<tr>
<td>9</td>
<td>Maximum active session limit reached. No more sessions are allowed. (See API Administration options Section 1.10)</td>
</tr>
<tr>
<td>10</td>
<td>An error was encountered while connecting to a server resource. (Also applicable for logout method)</td>
</tr>
<tr>
<td>11 &amp; 12</td>
<td>User input validation errors. Returned under any of the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• An error occurred while parsing user input.</td>
</tr>
<tr>
<td></td>
<td>• Heartbeat tag missing</td>
</tr>
</tbody>
</table>

URL:
https://csm-clienthost/heartbeatCallback

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:
```xml
<?xml version="1.0" encoding="UTF-8"?>
<heartbeatCallbackRequest>
  <protVersion>1.0</protVersion>
</heartbeatCallbackRequest>
```
Invalid callback URL specified or Callback URL is missing
• Callback URL specified when heartbeat requested is false

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Session Creation failed</td>
</tr>
<tr>
<td>16</td>
<td>Only HTTPS protocol allowed for callback url</td>
</tr>
</tbody>
</table>

### Table 14: Login Method Error Codes

#### 2.2.3 Method logout
The logout method notifies the CSM server that a previously authenticated CSM client is no longer requiring session access to the CSM server. The CSM client should logout from the CSM server if it does not intend to access methods on the CSM server within the session access timeout window.

#### 2.2.3.1 Request
An example of the method logout request is shown in the figure below. The fields in these messages are described in the table below.

**URL:**
https://hostname/nbi/logout

**HTTP Header:**
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

**XML Argument:**
```xml
<?xml version="1.0" encoding="UTF-8"?>
<logoutRequest>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
</logoutRequest>
```

**Figure 15: logout Request Example**

### Table 15: logout Request Elements and Attributes Descriptions

<table>
<thead>
<tr>
<th>HTTP/XML Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>logoutRequest</td>
<td>XML argument that logs out the CSM client from the CSM server</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>
2.2.3.2 Response

An example of the logout response is shown in the figure below. The fields in these messages are described in the table below.

![XML Response Example](image)

**Table 16: logout Response Elements and Attributes Description**

<table>
<thead>
<tr>
<th>XML Response</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>logoutResponse</td>
<td>Returns the session information or failure</td>
</tr>
</tbody>
</table>

![XML Response Example](image)

2.2.4 Method: ping

The ping method maintains an active authenticated session from timing out or being discarded by the server. The CSM client should call the ping method before every sessionTimeoutInMins to ensure the CSM server does not discard its authenticated session. Note: The authenticated session inactivity timeout is reset implicitly if an authentication session is used to make a call to the server.
2.2.4.1 Request
An example of the ping request is shown in the figure below. The fields in these messages are described in the table below.

URL:
https://hostname/nbi/ping

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:
<?xml version="1.0" encoding="UTF-8"?>
<pingRequest>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
</pingRequest>

Figure 19: ping Request Example

Table 17: ping Request Elements and Attributes Descriptions

<table>
<thead>
<tr>
<th>HTTP/XML Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pingRequest</td>
<td>XML argument that pings the CSM server</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>PUT</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

<xs:element name="pingRequest" type="PingRequest"/>
<xs:complexType name="PingRequest">
  <xs:complexContent>
    <xs:extension base="BaseReqResp"/>
  </xs:complexContent>
</xs:complexType>

Figure 20: PingRequest XML Schema

2.2.4.2 Response
An example of the ping response is shown in the figure below. The fields in these messages are described in the table below.
Figure 21: ping Response Example

Table 18: ping Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>XML Response</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pingResponse</td>
<td>Returns the ping information</td>
</tr>
</tbody>
</table>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pingResponse>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
</pingResponse>
```

Figure 22: PingResponse XML Schema

```xml
<xs:element name="pingResponse" type="PingResponse"/>
<xs:complexType name="PingResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:extension base="BaseReqResp"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```
3 CSM Configuration Service API

The configuration service provides access to read the configuration of network and CSM policy objects. The state of a policy may either be committed or deployed. It is possible that a committed policy is one that has not yet been deployed to a device. The configuration service API policy methods will return committed policies only. The policy commit and deployment to the device are two separate operations. It is possible that the current running device configuration does not reflect all policy configuration changes until they are deployed successfully. See the class definitions of BasePolicy and BasePolicyObject for details on how the configuration state is captured.

The configState attribute for BasePolicy and BasePolicyObject elements (section 3.1.1 and 3.1.2) indicates this state. If the config state is “committed” the device for which this config has been fetched has pending committed changes that are not yet deployed. If the config state is “deployed” then it means that all committed changes have been deployed i.e. the CSM policy and the device are in sync.

Selective Policy Management is a CSM feature that allows a CSM administrator to selectively manage policies in CSM. Data corresponding to any policy not selected for management will not be maintained in the CSM policy database. And such policy data will not be returned by this API. Please consider using the Utility Service API under such cases.

3.1 Object Model

The following sections describe the object model used by the CSM Configuration Service.

3.1.1 Base Policy

The CSM object model consists of two primary classes - a “Policy” class that denotes a specific policy such as AAA policy, Interface Policy, Firewall Access Rule etc. Policy classes can additionally reference “Policy Objects” (also called as Building Blocks) that are used to denote reusable objects such as network addresses, services, port lists etc. As an example, a “Firewall Policy” can define source and destinations addresses as a “Network Policy Objects”. Once defined policy objects are reusable and could be used by multiple policies.

All Policy classes extend from a “BasePolicy” class and all “Policy Objects” extend from a base class called as “BasePolicyObject”. The following class diagram denotes this relationship.
Figure 23: BasePolicy Class Inheritance
The BasePolicy class inherits from the BaseObject class including all attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>It's a mandatory attribute of any policy that describes content of the policy. Example – “AAA Policy”.</td>
</tr>
<tr>
<td>isMandatoryAggregation</td>
<td>boolean</td>
<td>Relevant only for onion aggregation. Returns whether a policy is mandatory or not: mandatory policies will be prepended to parent and default ones will be appended.</td>
</tr>
<tr>
<td>orderId</td>
<td>Int</td>
<td>O-based ordering id. Returns index of the policy in the policy record.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Policy Description (optional)</td>
</tr>
<tr>
<td>configState</td>
<td>Enumeration</td>
<td>The current state of the policy taken from { committed, deployed }</td>
</tr>
</tbody>
</table>

Table 19: BasePolicy Class Attributes

```
<xs:complexType name="BasePolicy">
  <xs:complexContent>
    <xs:extension base="BaseObject">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="type" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="orderId" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
        <xs:element name="isMandatoryAggregation" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 24: BasePolicy XML Schema

The following sections denote the data content for important policy and policy object types.
3.1.2 BasePolicyObject

This is the base class for all Policy Objects which are reusable object definitions. Policy data like AAA policy, Firewall policy etc. maintain references to policy objects. There are multiple “types” of Policy Objects. Also a policy object can be “Global” or “Local”. A global Policy object indicates that the object is a global definition that is referenced by policies on any device. A Policy Object “override” indicates a global policy object that is “overridden” for a specific device.

A set of policy objects may be “grouped” under a single policy object of the same “type”. In some cases, a policy object could also reference a policy object of a totally different type (this is different from a ‘grouped” policy object which groups’ policy objects of the same type).

The BasePolicyObject class inherits from BaseObject including all attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>It's a mandatory attribute of any policy object that describes the type of the policy object. Example – “Network” or “Service”.</td>
</tr>
<tr>
<td>comment</td>
<td>String</td>
<td>Associated comment/description of this policy object (optional).</td>
</tr>
<tr>
<td>nodeGID</td>
<td>Object Identifier</td>
<td>The device ID if this is a “override” policy object. Set to -1 for Globals</td>
</tr>
<tr>
<td>isProperty</td>
<td>Boolean</td>
<td>A true value indicates that “overrides” for this Global Policy Object are allowed. A false indicates this global is not allowed to be “overridden”</td>
</tr>
<tr>
<td>subType</td>
<td>String</td>
<td>Sub-types applicable for “Network” and “Service” Policy Objects. For example “Host”, “Address Range” are sub-types for a “Network” Policy object.</td>
</tr>
<tr>
<td>isGroup</td>
<td>Boolean</td>
<td>If true, this indicates whether this policy object is a “grouping” of other policy objects of the same type.</td>
</tr>
<tr>
<td>refGIDs</td>
<td>ObjectIdentifierList</td>
<td>Only applicable if “isGroup” is true. The list has the policy objects Id’s this refers to.</td>
</tr>
<tr>
<td>configState</td>
<td>Enumeration</td>
<td>The current state of the policy object taken from { committed, deployed }</td>
</tr>
</tbody>
</table>

Table 20: BasePolicyObject Class Definition

Name override behavior - This is the name associated with the object. If the name is empty (“”) then this refers to a internal policy object. All user defined policy objects must have a name. Internal policy objects are automatically created by the system in some cases. For example if a user provides a literal IP address in a rule (instead of a Policy Object), then a ‘nameless’ policy object is automatically created for the rule.

parentId = The parent Global Policy Object ID for which this Policy Object is an “override”. For non-overrides, this is set to -1.
Figure 25: BasePolicyObject XML Schema
3.1.3 Policy Utility Classes
The following utility classes are defined for use across multiple policies:

- NetworkInterfaceObjectsRefs
  - Refers to a list of networks, interfaceRoles and ipv4 data strings

- NetworkObjectsRefs
  - Refers to a list of networks and ipv4 data strings

- IdentityUserGrpObjectsRefs
  - Refers to a list of identity user group objects GIDs, user names and user groups

- SecurityGrpObjectsRef
  - Refers to a security name, security tag or a security object GID

- SecurityGrpObjectsRefs
  - Refers to a list of SecurityGrpObjectsRef

- NetworkObjectRefs
  - Refers to a network, ipv4 data string and interface keyword

- NetworkOrIPRef
  - Refers to either a host, network or a ipv4 data string
Figure 26: Policy Utility Class XML Schema
3.1.4 PolicyObject Derived Classes

This section and sub-sections define the supported PolicyObject classes for this API.

3.1.4.1 NetworkPolicyObject

A NetworkPolicyObject extends from the BasePolicyObject class and inherits all its attributes. A NetworkPolicyObject defines an IPv4 address, network or range.

Policy definitions reference the NetworkPolicyObject via the gid value. The inherited "subtype" attribute defines the type of IPv4 data contained. The allowable values for subtype for a NetworkPolicyObject are “Host”, “Network”, “Address Range”, “FQDN” and “Group”. The contents of a NetworkPolicyObject can also be “empty” in some cases when the inherited isGroup attribute is set to true (and subtype is “Group”). In such cases the NetworkPolicyObject is itself a container reference to “other Network Policy Objects”.

The list of gid values for such a PolicyObject is obtained from the refs inherited attribute. Also a “Group” NetworkPolicyObject can sometimes also contain multiple IPv4Data elements denoting literal IPv4 address, network or ranges. The combinations of data from the refs attribute references and the IPv4Data elements denote the complete group of addresses the policy object references.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv4Data</td>
<td>String</td>
<td>Defines a specific IPv4 data like address, range or network.</td>
</tr>
<tr>
<td>ipData</td>
<td>String</td>
<td>Defines a specific IP data like address, range or network. It can be both IPv4 and IPv6.</td>
</tr>
<tr>
<td>fqdnData</td>
<td>Complex</td>
<td>Contains Fully Qualified Domain Name (FQDN) if this is a FQDN type NetworkPolicyObject.</td>
</tr>
<tr>
<td>fqdnData.value</td>
<td>String</td>
<td>The FQDN string</td>
</tr>
<tr>
<td>fqdnData.isIPv4Only</td>
<td>boolean</td>
<td>If true, the command generated and sent to the device contains the “v4” parameter.</td>
</tr>
</tbody>
</table>

Table 21: NetworkPolicyObject Class Definition

NOTE: From API version 1.1, a new tag called <ipData> has been added to the Network Object definition. Network objects referenced in legacy policies like DeviceAccessRuleFirewallPolicy will continue to use <ipv4Data> as these policies only reference IPv4 addresses. However, newer policies like DeviceAccessRuleUnifiedFirewallPolicy will use the <ipData> tag in the policy. This is because an <ipData> tag can contain both IPv4 and IPv6 addresses.
3.1.4.2 IdentityUserGroupPolicyObject


<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>userNameData</td>
<td>String</td>
<td>List of users in the group object.</td>
</tr>
<tr>
<td>userGroupData</td>
<td>String</td>
<td>List of user groups part of this group object.</td>
</tr>
</tbody>
</table>

Table 22: IdentityUserGroupPolicyObject Class Definition
Figure 28: IdentityUserGroupPolicyObject XML Schema
3.1.4.3 PortListPolicyObject

A PortListPolicyObject extends from the BasePolicyObject class and inherits all its attributes. A PortListPolicyObject defines an individual port or a group of ports. Policy definitions reference the PortListPolicyObject via the gid value. The contents of a PortListPolicyObject can also be “empty” in some cases when the inherited isGroup attribute is set to true. In such cases the PortListPolicyObject is itself a container reference to “other Port List Policy Objects”. The list of gid values for such a PolicyObject is obtained from the refs inherited attribute.

In some cases the PortListPolicyObject can have a group of gid values from the refs attribute and additionally contain a definition for a single port list object. In such cases, the combination of references and the value inside the PortListPolicyObject element together constitute the groups of ports specified by the definition.

The following table defines the contents of the ServicePolicyObject:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPort</td>
<td>PortIdentifier</td>
<td>Defines the starting port.</td>
</tr>
<tr>
<td>endPort</td>
<td>PortIdentifier</td>
<td>Defines the end port. Start port to end port specify the range of ports specified by this PolicyObject. If start and end Ports are equal then the definition corresponds to a “single” port.</td>
</tr>
</tbody>
</table>

Table 23: PortListPolicyObject Class Definition

```xml
<xs:complexType name="PortListPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="startPort" type="PortIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="endPort" type="PortIdentifier" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 29: PortListPolicyObject XML Schema
3.1.4.4 ServicePolicyObject

A ServicePolicyObject extends from the **BasePolicyObject** class and inherits all its attributes. A ServicePolicyObject defines services such as tcp, udp etc. Policy definitions reference the ServicePolicyObject via the gid value. The contents of a ServicePolicyObject can also be “empty” in some cases when the inherited **isGroup** attribute is set to true. In such cases the ServicePolicyObject is itself a container reference to “other Service Policy Objects”. The list of gid values for such a PolicyObject is obtained from the **refs** inherited attribute.

In some cases the ServicePolicyObject can have a group of gid values from the **refs** attribute and additionally contain a definition for a single service. In such cases, the combination of references and the value inside the ServicePolicyObject element together constitute the groups of ports specified by the definition.

The following table defines the contents of the ServicePolicyObject:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>String</td>
<td>A string defining a protocol such as tcp, udp etc.</td>
</tr>
<tr>
<td>sourcePort</td>
<td>Complex</td>
<td>Element container that holds the source port data. This is a choice type complex type so only one of the sub-elements will be defined.</td>
</tr>
<tr>
<td>sourcePort.port</td>
<td>Unsigned Int</td>
<td>The actual port value.</td>
</tr>
<tr>
<td>sourcePort.portRef</td>
<td>ObjectIdentifier</td>
<td>Gid reference to a PortListPolicyObject.</td>
</tr>
<tr>
<td>destinationPort</td>
<td>Complex</td>
<td>Element container that holds the destination port data. This is a choice type complex type so only one of the sub-elements will be defined.</td>
</tr>
<tr>
<td>destinationPort.port</td>
<td>Unsigned Int</td>
<td>The actual port value.</td>
</tr>
<tr>
<td>destinationPort.portRef</td>
<td>ObjectIdentifier</td>
<td>Gid reference to a PortListPolicyObject.</td>
</tr>
<tr>
<td>icmpMessage</td>
<td>String</td>
<td>ICMP Message content.</td>
</tr>
</tbody>
</table>

Table 24: ServicePolicyObject Class Definition
Figure 30: ServicePolicyObject XML Schema
3.1.4.5 InterfaceRolePolicyObject

An InterfaceRolePolicyObject extends from the BasePolicyObject class and inherits all its attributes. A InterfaceRolePolicyObject defines a pattern to denote an interface or group of interfaces like FastEthernet0, inside, outside etc.. The inherited gid attribute specifies a unique ID for an ‘instance’ of a InterfaceRole PolicyObject. Policy definitions reference the InterfaceRolePolicyObject via the gid value.

The following table defines the contents of the InterfaceRolePolicyObject:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>String</td>
<td>Defines one or more interface role regex patterns. A&quot;*&quot; will match all interfaces.</td>
</tr>
</tbody>
</table>

Table 25: InterfaceRolePolicyObject Class Definition

```xml
<xs:complexType name="InterfaceRolePolicyObject">
    <xs:complexContent>
        <xs:extension base="BasePolicyObject">
            <xs:sequence minOccurs="1" maxOccurs="1">
                <xs:element name="pattern" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Figure 31: InterfaceRolePolicyObject XML Schema
3.1.4.6 TimeRangePolicyObject

A TimeRangePolicyObject extends from the BasePolicyObject class and inherits all its attributes. A TimeRangePolicyObject defines a time range and its associated recurrence. Policy definitions reference the TimeRangePolicyObject via the gid value. The contents of a TimeRangePolicyObject cannot be empty and there are also no “grouping” for TimeRangePolicyObject supported.

The following table defines the contents of the TimeRangePolicyObject:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>dateTime</td>
<td>Specifies the start time. If the start time is not specified, it means the time range specified by this range is “already started”.</td>
</tr>
<tr>
<td>endTime</td>
<td>dateTime</td>
<td>Specifies the end time. If the end time is not specified, it means the time range specified by this range is “never ends”.</td>
</tr>
<tr>
<td>recurrence</td>
<td>Complex type</td>
<td>A time range can have zero or more recurring ranges that qualify the recurrence for the time range. This is a choice type complex element and can contain either a “Day of the Week” based recurrence pattern or “Weekly” based recurrence pattern</td>
</tr>
<tr>
<td>recurrence.dayOfWeekInterval</td>
<td>Complex Type</td>
<td>Defines the “Day of the Week” based recurrence.</td>
</tr>
<tr>
<td>recurrence.dayOfWeekInterval.dayOfWeek</td>
<td>String</td>
<td>The dayOfWeek attribute defines days of the week that the pattern should be applicable. It can have any “one” of the following String values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– <strong>Everyday</strong>: For every day recurrence (OR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– <strong>Weekday</strong>: For Week day recurrence (OR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– <strong>Weekends</strong>: For Week end recurrence (OR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– A comma separated list of specific days of the week – Monday, Tuesday, .. etc.</td>
</tr>
<tr>
<td>recurrence.dayOfWeekInterval.startTime</td>
<td>Time</td>
<td>Specifies the start time.</td>
</tr>
<tr>
<td>recurrence.dayOfWeekInterval.endTime</td>
<td>Time</td>
<td>Specifies the end time.</td>
</tr>
<tr>
<td>recurrence.weeklyInterval</td>
<td>Complex Type</td>
<td>Defines the “Weekly” based recurrence.</td>
</tr>
<tr>
<td>recurrence.weeklyInterval.startDay</td>
<td>String</td>
<td>Defines the start day i.e “Monday”, “Tuesday” ….</td>
</tr>
<tr>
<td>recurrence.weeklyInterval.startTime</td>
<td>Time</td>
<td>Specifies the start time.</td>
</tr>
<tr>
<td>recurrence.weeklyInterval.endDay</td>
<td>String</td>
<td>Defines the end day i.e “Monday”, “Tuesday” ….</td>
</tr>
<tr>
<td>recurrence.weeklyInterval.endTime</td>
<td>Time</td>
<td>Specifies the end time.</td>
</tr>
</tbody>
</table>

Table 26: TimeRangePolicyObject Class Definition
Figure 32: TimeRangePolicyObject XML Schema
3.1.4.7 SLA Monitor Policy Object


The following table defines the contents of the SLAMonitorPolicyObject:

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>slaId</td>
<td>int</td>
<td>The ID number of the SLA operation.</td>
</tr>
<tr>
<td>interface</td>
<td>ObjectIdentifier</td>
<td>The source interface for all ICMP echo requests sent to the monitored address to test its availability. Enter the name of an interface or interface role, or click Select to select an it from a list or to create a new interface role.</td>
</tr>
<tr>
<td>monitoredAddress</td>
<td>string</td>
<td>The IP address that is being monitored for availability by the SLA operation.</td>
</tr>
<tr>
<td>dataSizeInBytes</td>
<td>int</td>
<td>The size of the ICMP request packet payload, in bytes.</td>
</tr>
<tr>
<td>thresholdInMilliSeconds</td>
<td>int</td>
<td>The amount of time that must pass after an ICMP echo request before a rising threshold is declared, in milliseconds.</td>
</tr>
<tr>
<td>timeoutInMilliSeconds</td>
<td>int</td>
<td>The amount of time that the SLA operation waits for a response to the ICMP echo requests, in milliseconds.</td>
</tr>
<tr>
<td>frequencyInSeconds</td>
<td>int</td>
<td>The frequency of ICMP echo request transmissions, in seconds.</td>
</tr>
<tr>
<td>toS</td>
<td>int</td>
<td>The type of service (ToS) defined in the IP header of the ICMP request packet. Values range from 0 to 255. The default is 0.</td>
</tr>
<tr>
<td>numberOfPackets</td>
<td>int</td>
<td>The number of packets that are sent. Values range from 1 to 100. The default is 1 packet.</td>
</tr>
</tbody>
</table>

Table 27: SLAMonitorPolicyObject Class Definition
Figure 33: SLAMonitorPolicyObject XML Schema
### 3.1.4.8 Standard ACE Policy Object

A StandardACEPolicyObject extends from the **BasePolicyObject** class and inherits all its attributes. A StandardACEPolicyObject defines a standard IP access control entry. Policy definitions reference the StandardACEPolicyObject via the gid value.

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>networkGID</td>
<td>ObjectIdentifier</td>
<td>The source or destination of the traffic.</td>
</tr>
<tr>
<td>doLogging</td>
<td>boolean</td>
<td>Whether to create log entries when traffic meets the entry criteria.</td>
</tr>
<tr>
<td>permit</td>
<td>boolean</td>
<td>Indicates the action to be taken when a match is found</td>
</tr>
</tbody>
</table>

Table 28: StandardACEPolicyObject Class Definition

```xml
<xs:complexType name="StandardACEPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="networkGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="doLogging" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
        <xs:element name="permit" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 34: StandardACEPolicyObject XML Schema

### 3.1.4.9 Extended ACE Policy Object

An ExtendedACEPolicyObject extends from the **BasePolicyObject** class and inherits all its attributes. An ExtendedACEPolicyObject defines an extended access control entry. Policy definitions reference the ExtendedACEPolicyObject via the gid value.

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceGID</td>
<td>ObjectIdentifier</td>
<td>The source of the traffic.</td>
</tr>
<tr>
<td>destinationGID</td>
<td>ObjectIdentifier</td>
<td>Traffic destination.</td>
</tr>
<tr>
<td>serviceGID</td>
<td>ObjectIdentifier</td>
<td>The service that defines the type of traffic to act upon</td>
</tr>
<tr>
<td>doLogging</td>
<td>String</td>
<td>Contains value “true” if logging is enabled for PIX, ASA, FWSM devices or “false” otherwise. If logInterval and logLevel elements are not specified then it means “Default Logging” is enabled.</td>
</tr>
</tbody>
</table>
### logInterval

*String*

Specifies the Logging Interval in seconds, if this is specified it means, “per ACE Logging is Enabled”

### logLevel

*String*

Specifies the Logging Level – one of “Emergency”, “Alert”, “Critical”, “Error”, “Warning”, “Notification”, “Informational” or “Debugging. If this is specified it means, “per ACE Logging is Enabled”

### logOption

*String*

Used to specify IOS logging. Contains “log” if IOS logging is enabled. Contains “log-input” if IOS Logging is enabled and Log Input is also enabled for IOS devices.

### permit

*boolean*

True if this is a permit ACE, false for deny.

---

**Table 29: ExtendedACEPolicyObject Class Definition**

```xml
<xs:complexType name="ExtendedACEPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence>
        <xs:element name="sourceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="destinationGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="serviceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="doLogging" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="logInterval" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="logLevel" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="logOption" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="permit" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

---

**Figure 35: ExtendedACEPolicyObject XML Schema**

---

Cisco Security Manager 4.4 API Specification (Version 1.1)

OL- 29074-01
3.1.4.10 ACL Policy Object

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>references</td>
<td>Complex Type</td>
<td>List of references to ACE’s</td>
</tr>
<tr>
<td>references.sequenceNumber</td>
<td>Unsigned int</td>
<td>Sequence number of this entry</td>
</tr>
<tr>
<td>references.aclObjectReferenceGID</td>
<td>ObjectIdentifier</td>
<td>Reference to the ACE policy object</td>
</tr>
<tr>
<td>References.aceReferenceGID</td>
<td>ObjectIdentifier</td>
<td>Reference to the standard/extended ACE policy object</td>
</tr>
</tbody>
</table>

Table 30: ACLPolicyObject Class Definition

```xml
<xs:complexType name="ACLPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="references" minOccurs="1" maxOccurs="unbounded">
          <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="sequenceNumber" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
            <xs:choice>
              <xs:element name="aclObjectReferenceGID" type="ObjectIdentifier"/>
              <xs:element name="aceReferenceGID" type="ObjectIdentifier"/>
            </xs:choice>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 36: ACLPolicyObject XML Schema

3.1.4.11 SecurityGroupPolicyObject

A SecurityGroupPolicyObject extends from the BasePolicyObject class and inherits all its attributes. A SecurityGroupPolicyObject defines a security tag or name.

Policy definitions reference the SecurityGroupPolicyObject via the gid value. The contents of a SecurityGroupPolicyObject can also be “empty” in some cases when the inherited isGroup attribute is set to true. In such cases the SecurityGroupPolicyObject is itself a container reference to “other Security Group Policy Objects”.

The list of gid values for such a PolicyObject can be obtained from the refs inherited attribute. Also a “Group” SecurityGroupPolicyObject can sometimes also contain multiple securTag elements denoting literal security.
tag/name or contained policy object. The complete content of the security policy object can be obtained from the securityTag elements.

This object is available from the version 1.1 of the API.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>securityTag</td>
<td>Complex(SecurityGrpObjectsRef)</td>
<td>Defines a security tag/name or GID of the referred policy object.</td>
</tr>
<tr>
<td>securityTag</td>
<td>ObjectIdentifier</td>
<td>Defines the GID of the referred security group policy object.</td>
</tr>
<tr>
<td>securityTag</td>
<td>String</td>
<td>Defines a security tag/</td>
</tr>
<tr>
<td>securityTag</td>
<td>String</td>
<td>Defines a security tag/name</td>
</tr>
</tbody>
</table>

Table 31: SecurityGroupPolicyObject() Class Definition

```xml
<xs:complexType name="SecurityGroupPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence>
        <xs:element name="securityTag" type="SecurityGrpObjectsRef" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 37: SecurityGroupObject XML Schema

```xml
<xs:complexType name="SecurityGrpObjectsRef">
  <xs:sequence>
    <xs:choice>
      <xs:element name="securityGrpObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
      <xs:element name="secName" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="secTag" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Figure 38: SecurityGrpObjectsRef XML Schema
3.1.5 Policy Derived Classes

This section and the following sub-sections define supported Policy classes over this API.

3.1.5.1 DeviceAccessRuleFirewallPolicy

A DeviceAccessRuleFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a DeviceAccessRuleFirewallPolicy denotes a single access control entry. The orderId attribute from the BasePolicy class defines the ordering of these rules.


The sources and destination elements may contain a combination of any of the following:

- networkObjectGIDs: This includes one or more GID references to a Network Policy Object
- interfaceRoleObjectGIDs: This includes one or more GID references to a Interface Role Policy Object
- ipv4Data: One or more literal IPv4 addresses

It is possible to specify a destination element referring two Network Policy Objects and also including a literal address such as 1.1.1.1/32.

The interfaceRoleObjectIDs specified outside the sources and destination sub-elements specifies the interface on which the ACL is applied.

The following table defines the contents of a DeviceAccessRuleFirewallPolicy:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEnabled</td>
<td>Boolean</td>
<td>True if the rule is enabled, false otherwise.</td>
</tr>
<tr>
<td>direction</td>
<td>String</td>
<td>In or out.</td>
</tr>
<tr>
<td>permit</td>
<td>boolean</td>
<td>True indicates a permit and false indicates a deny</td>
</tr>
<tr>
<td>interfaceRoleObjectIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of InterfaceRole Policy Objects. The ID links to the gid attribute of the corresponding InterfaceRole object</td>
</tr>
<tr>
<td>users</td>
<td>ComplexType</td>
<td>Contain user and user groups for whom the rule is applicable (applicable only for ASA device versions greater than or equal to 8.4(2))</td>
</tr>
<tr>
<td>users. identityUserGrpObject GIDs</td>
<td>ObjectIdentifierList</td>
<td>Reference to list of IdentityUserGroupPolicyObject object GIDs.</td>
</tr>
<tr>
<td>users. userNameData</td>
<td>String</td>
<td>List of users.</td>
</tr>
<tr>
<td>users. userGroupData</td>
<td>String</td>
<td>List of user groups.</td>
</tr>
<tr>
<td>sources</td>
<td>ComplexType</td>
<td>Container for source network and interface roles.</td>
</tr>
<tr>
<td>sources.networkObject GIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of Network Policy Objects. The ID links to the gid attribute of the corresponding Network object</td>
</tr>
</tbody>
</table>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sources.interfaceRoleObjectGIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of InterfaceRole Policy Objects. The ID links to the gid attribute of the corresponding InterfaceRole object.</td>
</tr>
<tr>
<td>sources.ipv4Data</td>
<td>String</td>
<td>Multiple IPv4Data elements containing either IPv4 host, network or ranges.</td>
</tr>
<tr>
<td>destination</td>
<td>ComplexType</td>
<td>Container for destination network and interface roles.</td>
</tr>
<tr>
<td>destination.networkObjectGIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of Network Policy Objects. The ID links to the gid attribute of the corresponding Network object.</td>
</tr>
<tr>
<td>destination.interfaceObjectGIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of InterfaceRole Policy Objects. The ID links to the gid attribute of the corresponding InterfaceRole object.</td>
</tr>
<tr>
<td>destination.ipv4Data</td>
<td>String</td>
<td>Multiple IPv4Data elements containing either IPv4 host, network or ranges.</td>
</tr>
<tr>
<td>serviceObjectIDs</td>
<td>ObjectIdentifierList</td>
<td>A list of ObjectIdentifier IDs that reference a set of Service Policy Objects. The ID links to the gid attribute of the corresponding Service object.</td>
</tr>
<tr>
<td>serviceParameters</td>
<td>ServiceParameters</td>
<td>Multiple service data elements that can store service data including protocol and port details.</td>
</tr>
<tr>
<td>serviceParameters.protocol</td>
<td>String</td>
<td>A string defining a protocol such as tcp, udp etc.</td>
</tr>
<tr>
<td>serviceParameters.sourcePort</td>
<td>ComplexType</td>
<td>Element container that holds the source port data. This is a choice type complex type so only one of the sub-elements will be defined.</td>
</tr>
<tr>
<td>serviceParameters.sourcePort.port</td>
<td>String</td>
<td>The actual port value. Can also be a range specified in the form &lt;start port&gt;-&lt;end port&gt; example: 1-65535</td>
</tr>
<tr>
<td>serviceParameters.sourcePort.portRef</td>
<td>ObjectIdentifier</td>
<td>Gid reference to a PortListPolicyObject.</td>
</tr>
<tr>
<td>serviceParameters.destinationPort</td>
<td>ComplexType</td>
<td>Element container that holds the destination port data. This is a choice type complex type so only one of the sub-elements will be defined.</td>
</tr>
<tr>
<td>serviceParameters.destinationPort.port</td>
<td>String</td>
<td>The actual port value. Can also be a range specified in the form &lt;start port&gt;-&lt;end port&gt; example: 1-65535</td>
</tr>
<tr>
<td>serviceParameters.destinationPort.portRef</td>
<td>ObjectIdentifier</td>
<td>Gid reference to a PortListPolicyObject.</td>
</tr>
<tr>
<td>serviceParameters.icmpMessage</td>
<td>String</td>
<td>ICMP Message content.</td>
</tr>
<tr>
<td>logOptions</td>
<td>ComplexType</td>
<td>Root element that stores the logging options associated with a rule.</td>
</tr>
<tr>
<td>logOptions.isFirewallLog</td>
<td>Boolean</td>
<td>Indicates whether logging is enabled for PIX, ASA, FWSM rules.</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ggingEnabled</td>
<td>Boolean</td>
<td>Whether default logging is enabled. This is applicable for PIX, ASA and FWSM devices and only if isFirewallLoggingEnabled is true</td>
</tr>
<tr>
<td>logOptions.isDefaultLogging</td>
<td>Boolean</td>
<td>Specifies the logging interval in seconds. This is applicable for PIX, FWSM and ASA and if isDefaultLogging is false or not specified. (Applicable for per ACE logging)</td>
</tr>
<tr>
<td>logOptions.loggingInterval</td>
<td>Int</td>
<td>Specifies a logging level – “Emergency”, “Alert”, “Critical”, “Error”, “Warning”, “Notification”, “Informational” or “Debugging”. This is applicable for PIX, FWSM and ASA and if isDefaultLogging is false or not specified. (Applicable for per ACE logging)</td>
</tr>
<tr>
<td>logOptions.loggingLevel</td>
<td>String</td>
<td>Set to true is logging for IOS devices is enabled. Only applicable for IOS</td>
</tr>
<tr>
<td>logOptions.isLogInput</td>
<td>Boolean</td>
<td>True if log input is enabled. This is applicable for IOS devices and only if isIOSLoggingEnabled is true.</td>
</tr>
<tr>
<td>iosOptions</td>
<td>String</td>
<td>IOS Options allowed values are “none”, “Fragment” and “Established”.</td>
</tr>
<tr>
<td>timeRangeObjectId</td>
<td>ObjectIdentifier</td>
<td>References a TimeRangePolicyObject if a time range is specified for this rule.</td>
</tr>
</tbody>
</table>

**Table 32: DeviceAccessRuleFirewallPolicy Class Definition**

See the XML schema at the end of this document for the XML schema for this class.
3.1.5.1.1 Policy Config Device Response Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<n:policyConfigDeviceResponse xmlns:n="csm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <policy>
    <deviceAccessRuleFirewallPolicy>
      <gid>00000000-0000-0000-0000-000000000889</gid>
      <name>string</name>
      <lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
      <updatedByUser>admin</updatedByUser>
      <lastCommitTime>2012-12-05T09:39:371Z</lastCommitTime>
      <activityName>admin_05_Dec.2012_01.17.10</activityName>
      <type>Access Rule</type>
      <orderId>0</orderId>
      <isMandatoryAggregation>false</isMandatoryAggregation>
      <description/>
      <isEnabled>true</isEnabled>
      <direction>In</direction>
      <permit>true</permit>
      <interfaceRoleObjectGIDs>00000000-0000-0000-0000-000000000123</interfaceRoleObjectGIDs>
      <sources>
        <networkObjectGIDs>00000000-0000-0000-0000-000000000124</networkObjectGIDs>
        <interfaceRoleObjectGIDs>00000000-0000-0000-0000-000000000123</interfaceRoleObjectGIDs>
      </sources>
      <destinations>
        <networkObjectGIDs>00000000-0000-0000-0000-000000000125</networkObjectGIDs>
        <interfaceRoleObjectGIDs>00000000-0000-0000-0000-000000000123</interfaceRoleObjectGIDs>
      </destinations>
      <services>
        <serviceObjectGIDs>00000000-0000-0000-0000-000000000126</serviceObjectGIDs>
        <serviceParameters>
          <protocol>Tcp</protocol>
          <sourcePort>
            <port>80</port>
          </sourcePort>
          <destinationPort>
            <port>80</port>
          </destinationPort>
        </serviceParameters>
      </services>
      <logOptions>
        <isFirewallLoggingEnabled>false</isFirewallLoggingEnabled>
      </logOptions>
    </deviceAccessRuleFirewallPolicy>
  </policy>
</n:policyConfigDeviceResponse>
```
<iosOptions>None</iosOptions>
<timeRangeObjectGID>00000000-0000-0000-0000-000000000127</timeRangeObjectGID>
</deviceAccessRuleFirewallPolicy>
</policyObject>
<policyObject>
<networkPolicyObject>
<gid>00000000-0000-0000-0000-000000000124</gid>
<name>mySource</name>
<lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
?type>Network</type>
<comment/>
<isProperty>false</isProperty>
<subType>Host</subType>
<isGroup>false</isGroup>
<refGIDs/>
<ipv4Data>1.1.2.2</ipv4Data>
</networkPolicyObject>
<networkPolicyObject>
<gid>00000000-0000-0000-0000-000000000125</gid>
<name>myDest</name>
<lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
?type>Network</type>
<comment/>
<isProperty>false</isProperty>
<subType>Host</subType>
<isGroup>false</isGroup>
<refGIDs/>
<ipv4Data>1.1.3.3</ipv4Data>
</networkPolicyObject>
<portListPolicyObject>
<gid>00000000-0000-0000-0000-0000000002782</gid>
<name>MyPortList</name>
<lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
?type>PortList</type>
<comment/>
<isProperty>false</isProperty>
<subType>Host</subType>
<isGroup>false</isGroup>
<refGIDs/>
<imgPort>514</imgPort>
<endPort>514</endPort>
</portListPolicyObject>
<servicePolicyObject>
<gid>00000000-0000-0000-0000-000000000126</gid>
<name>string</name>
<lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
?type>Service</type>
<comment/>
<isProperty>false</isProperty>
<subType/>
<isGroup>false</isGroup>
<refGIDs/>
<serviceParameters>
  <protocol>udp</protocol>
  <sourcePort>
    <portRefGID>00000000-0000-0000-0000-0000000002782</portRefGID>
  </sourcePort>
  <destinationPort>
    <port>514</port>
  </destinationPort>
  <icmpMessage>string</icmpMessage>
</serviceParameters>
</servicePolicyObject>
<interfaceRolePolicyObject>
  <gid>00000000-0000-0000-0000-000000000123</gid>
  <name>FA1</name>
  <lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
  <type>InterfaceRole</type>
  <comment/>
  <isProperty>false</isProperty>
  <subType/>
  <isGroup>false</isGroup>
  <refGIDs/>
  <pattern>FastEthernet0/0</pattern>
</interfaceRolePolicyObject>
<timeRangePolicyObject>
  <gid>00000000-0000-0000-0000-000000000127</gid>
  <name>string</name>
  <lastUpdateTime>1697-02-01T00:00:00Z</lastUpdateTime>
  <type>TimeRange</type>
  <comment/>
  <isProperty>false</isProperty>
  <subType/>
  <isGroup>false</isGroup>
  <refGIDs/>
  <startTime>1697-02-01T00:00:00Z</startTime>
  <endTime>1697-02-01T00:00:00Z</endTime>
  <recurrence>
    <weeklyInterval>
      <startDay>Monday</startDay>
      <startTime>13:20:00-05:00</startTime>
      <endDay>Friday</endDay>
      <endTime>13:20:00-05:00</endTime>
    </weeklyInterval>
  </recurrence>
</timeRangePolicyObject>
</policyObject>
</policyConfigDeviceResponse>
3.1.5.2 DeviceAccessRuleUnifiedFirewallPolicy

A DeviceAccessRuleUnifiedFirewallPolicy extends from the base DeviceAccessRuleFirewallPolicy class and inherits all its attributes. An instance of a DeviceAccessRuleUnifiedFirewallPolicy denotes a single unified access control entry. The orderId attribute from the BasePolicy class defines the ordering of these rules.

A DeviceAccessRuleUnifiedFirewallPolicy may additionally reference SecurityGroupPolicyObject,

The sourceSG and destinationSG elements SecurityGrpObjectsRefs which is a list of SecurityGrpObjectsRef. Refer to Sec 3.1.4.11 on SecurityGrpObjectsRef

The XML content contains all the attributes of the base except for IOS options like logging, fragment and established.

This policy is available starting from version 1.1 of the API.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceSG</td>
<td>Complex(SecurityGrpObjectsRefs)</td>
<td>Defines security tags or objects for the source of the incoming packet.</td>
</tr>
<tr>
<td>destinationSG</td>
<td>Complex(SecurityGrpObjectsRefs)</td>
<td>Defines security tags or objects for the destination of the incoming packet</td>
</tr>
</tbody>
</table>

Table 33: DeviceAccessRuleUnifiedFirewallPolicy Class Definition

```
<xs:complexType name="DeviceAccessRuleUnifiedFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="DeviceAccessRuleFirewallPolicy">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="sourceSG" type="SecurityGrpObjectsRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="destinationSG" type="SecurityGrpObjectsRefs" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 37: DeviceAccessRuleUnifiedFirewallPolicy XML Schema
### 3.1.5.3 DeviceStaticRoutingFirewallPolicy

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FwStaticRoutePolicy</td>
<td>Complex Type</td>
<td>Interface to which this static route applies.</td>
</tr>
<tr>
<td>interfaceGID</td>
<td>ObjectIdentifier</td>
<td>Destination network(s). You can provide one or more IP address/netmask entries, one or more Networks/Hosts objects, or a combination of both; separate the entries with commas. Use 0.0.0.0 to specify a default route. The 0.0.0.0 IP address can be abbreviated as 0.</td>
</tr>
<tr>
<td>networks</td>
<td>Complex Type</td>
<td>Value as reference to Policy Object</td>
</tr>
<tr>
<td>networks.ipv4Data</td>
<td>string</td>
<td>Value as raw string</td>
</tr>
<tr>
<td>gateway</td>
<td>Complex Type</td>
<td>The gateway router, which is the next hop for this route.</td>
</tr>
<tr>
<td>gateway.hostObjectIdentifier</td>
<td>ObjectIdentifier</td>
<td>Reference to Policy Object</td>
</tr>
<tr>
<td>gateway.ipv4Data</td>
<td>string</td>
<td>Raw string value for gateway IP address</td>
</tr>
<tr>
<td>metric</td>
<td>unsignedInt</td>
<td>The number of hops to the destination network. Valid values range from 1 to 255; the default value is 1.</td>
</tr>
<tr>
<td>tunnelled</td>
<td>boolean</td>
<td>Indicates whether this is a tunnel route.</td>
</tr>
<tr>
<td>slaMonitorGID</td>
<td>ObjectIdentifier</td>
<td>Name of an SLA (service level agreement) object that defines the monitoring policy.</td>
</tr>
</tbody>
</table>

**Table 34: DeviceStaticRoutingFirewallPolicy**
<xs:complexType name="DeviceStaticRoutingFirewallPolicy">
    <xs:complexContent>
        <xs:extension base="BasePolicy">
            <xs:sequence>
                <xs:element name="interfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
                <xs:element name="networks" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence minOccurs="1" maxOccurs="1">
                            <xs:element name="networkObjectIdentifierGIDs" type="ObjectIdentifierList" minOccurs="1" maxOccurs="1"/>
                            <xs:element name="ipv4Data" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="gateway" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence minOccurs="1" maxOccurs="1">
                            <xs:element name="hostObjectIdentifierGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
                            <xs:element name="ipv4Data" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="metric" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
                <xs:element name="tunnelled" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
                <xs:element name="slaMonitorGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

Figure 38: DeviceStaticRoutingFirewallPolicy
### 3.1.5.4 DeviceStaticRoutingRouterPolicy

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationNetwork</td>
<td>Complex Type</td>
<td>Container of elements for destination network configuration.</td>
</tr>
<tr>
<td>destinationNetwork.useAsDefaultRoute</td>
<td>boolean</td>
<td>Indicates whether the static route is the default route for unknown packets being forwarded by this router.</td>
</tr>
<tr>
<td>destinationNetwork.prefix</td>
<td>Complex Type</td>
<td>The destination IP address of the static route.</td>
</tr>
<tr>
<td>destinationNetwork.prefix.networkObjectReferenceGID</td>
<td>ObjectIdentifier</td>
<td>The destination IP address of the static route as Object Reference.</td>
</tr>
<tr>
<td>destinationNetwork.prefix.ipV4Data</td>
<td>string</td>
<td>The destination IP address of the static route in string format.</td>
</tr>
<tr>
<td>forwarding</td>
<td>Complex Type</td>
<td></td>
</tr>
<tr>
<td>forwarding.forwardingInterfaceGID</td>
<td>ObjectIdentifier</td>
<td>The interface name associated with the gateway router that is the next hop address for this router.</td>
</tr>
<tr>
<td>forwarding.forwardingIPAddress</td>
<td>Complex Type</td>
<td>The IP address associated with the gateway router that is the next hop address for this router.</td>
</tr>
<tr>
<td>forwarding.forwardingIPAddress.hostObjectReferenceGID</td>
<td>ObjectIdentifier</td>
<td>IP Address as object reference</td>
</tr>
<tr>
<td>forwarding.forwardingIPAddress.ipv4HostAddress</td>
<td>string</td>
<td>IP Address as string value</td>
</tr>
<tr>
<td>distanceMetric</td>
<td>unsignedInt</td>
<td>The number of hops from the gateway IP to the destination. The metric determines the priority of this route. The fewer the hops, the higher the priority assigned to the route, based on lower costs. When two routing entries specify the same network, the entry with the lower metric (that is, the higher priority) is selected.</td>
</tr>
<tr>
<td>isPermanentRoute</td>
<td>boolean</td>
<td>Indicates whether the static route is defined as a permanent route, which means that it will not be removed even if the interface is shut down or if the router is unable to communicate with the next router.</td>
</tr>
</tbody>
</table>

Table 35: DeviceStaticRoutingRouterPolicy
Figure 39: DeviceStaticRoutingRouterPolicy
### 3.1.5.5 DeviceBGPRouterPolicy

<table>
<thead>
<tr>
<th>Element/Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>asNumber</td>
<td>unsignedLong</td>
<td>The number of the autonomous system in which the router is located. Valid values range from 1 to 65535. This number enables a BGP routing process.</td>
</tr>
<tr>
<td>networks</td>
<td>Complex Type</td>
<td>The networks associated with the BGP route.</td>
</tr>
<tr>
<td>networks.networkObjectIdentifierGIDs</td>
<td>ObjectIdentifierList</td>
<td>Value as reference to Policy Object</td>
</tr>
<tr>
<td>networks.ipv4Data</td>
<td>string</td>
<td>Value as string</td>
</tr>
<tr>
<td>neighbors</td>
<td>Complex Type</td>
<td>The <em>internal</em> neighbors (those located in the same AS as the router) and <em>external</em> neighbors (located in different ASs) of the router.</td>
</tr>
<tr>
<td>neighbors.ipAddress</td>
<td>Complex Type</td>
<td>Neighorb address</td>
</tr>
<tr>
<td>neighbors.ipAddress.networkObjectGIDs</td>
<td>ObjectIdentifierList</td>
<td>Address as reference to policy object list</td>
</tr>
<tr>
<td>neighbors.ipAddress.ipv4Data</td>
<td>string</td>
<td>Address as string value</td>
</tr>
<tr>
<td>neighbors.asNumber</td>
<td>unsignedLong</td>
<td>AS number of the neighbor</td>
</tr>
<tr>
<td>autoSummary</td>
<td>boolean</td>
<td>Enables auto summary. When a subnet is redistributed from an IGP (such as RIP, OSPF or EIGRP) into BGP, this BGP version 3 features injects only the network route into the BGP table. Automatic summarization reduces the size and complexity of the routing table that the router must maintain.</td>
</tr>
<tr>
<td>synchronization</td>
<td>boolean</td>
<td>Enables Synchronization. When selected, synchronization is enabled. Use this feature to ensure that all routers in your network are consistent about the routes they advertise. Synchronization forces BGP to wait until the IGP propagates routing information across the AS. When deselected, synchronization is disabled. You can disable synchronization if this router does not pass traffic from a different AS to a third AS, or if all the routers in the AS are running BGP. Disabling this feature has the benefit of reducing the number of routes the IGP must carry, which improves convergence times. This is the default.</td>
</tr>
<tr>
<td>logNeighbor</td>
<td>boolean</td>
<td>Enables the logging of messages that are generated when a BGP neighbors resets, connects to the network, or is disconnected.</td>
</tr>
<tr>
<td>redistributionEntry</td>
<td>Complex Type</td>
<td>Redistribution settings when performing redistribution into a BGP autonomous system (AS). Can be multiple.</td>
</tr>
<tr>
<td>Method/Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>redistributionEntry.protocol</td>
<td>Complex Type</td>
<td>The protocol that is being redistributed.</td>
</tr>
<tr>
<td>redistributionEntry.protocol.static</td>
<td>String</td>
<td>Choice for protocol. Allowable values “IP” or “OSI”</td>
</tr>
<tr>
<td>redistributionEntry.protocol.connected</td>
<td>String</td>
<td>Choice for protocol</td>
</tr>
<tr>
<td>redistributionEntry.protocol.rip</td>
<td>String</td>
<td>Choice for protocol</td>
</tr>
<tr>
<td>redistributionEntry.protocol.eigrp</td>
<td>Complex Type</td>
<td>Choice for protocol</td>
</tr>
<tr>
<td>redistributionEntry.protocol.eigrp.asNumber</td>
<td>unsignedInt</td>
<td>The AS number if EIGRP is selected.</td>
</tr>
<tr>
<td>redistributionEntry.protocol.ospf</td>
<td>Complex Type</td>
<td>Choice of protocol</td>
</tr>
<tr>
<td>redistributionEntry.protocol.ospf.processId</td>
<td>UnsignedInt</td>
<td>The process number if EIGRP is selected.</td>
</tr>
<tr>
<td>redistributionEntry.protocol.ospf.match</td>
<td>String</td>
<td>Multiple Match criteria. Takes one of the allowable values for each match criteria:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Internal</strong>: Routes internal to the autonomous system (AS) are redistributed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>External1</strong>: Type 1 routes external to the AS are redistributed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>External2</strong>: Type 2 routes external to the AS are redistributed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>NSSAExternal1</strong>: Type 1 routes external to a not-so-stubby area (NSSA) are redistributed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>NSSAExternal2</strong>: Type 2 routes external to an NSSA are redistributed.</td>
</tr>
<tr>
<td>redistributionEntry.metric</td>
<td>unsignedLong</td>
<td>The value that determines the priority of the redistributed route.</td>
</tr>
</tbody>
</table>

**Table 36: DeviceBGPRouterPolicy Class Definition**

See the XML schema at the end of this document for the XML schema for this class.
3.1.5.6 InterfaceNATRouterPolicy

An InterfaceNATRouterPolicy extends from the BasePolicy class and inherits all its attributes. An instance of an InterfaceNATRouterPolicy specifies one NAT inside and outside interface.

The InterfaceNATRouterPolicy can reference an InterfaceRole PolicyObject.

The following table defines the contents of an InterfaceNATRouterPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>interfaceGID</td>
<td>ObjectIdentifier</td>
<td>An ObjectIdentifier ID that references an InterfaceRole Policy Objects denoting the inside or outside interfaces. The ID links to the gid attribute of the corresponding InterfaceRole object.</td>
</tr>
<tr>
<td>isNatInside</td>
<td>boolean</td>
<td>A boolean which indicates whether this is a NAT inside interface (true) or outside interface (false)</td>
</tr>
</tbody>
</table>

Table 37: InterfaceNATRouterPolicy Class Definition

```
<xs:complexType name="InterfaceNATRouterPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="interfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="isNatInside" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 39: InterfaceNATRouterPolicy XML Schema
### 3.1.5.7 InterfaceNATStaticRulesRouterPolicy

An InterfaceNATStaticRulesRouterPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of an InterfaceNATStaticRulesRouterPolicy denotes static NAT rules.

The InterfaceNATStaticRulesRouterPolicy can reference a Network Policy Object and an InterfaceRole Policy Object. The base orderId attribute defines the ordering of the static rules.

The following table defines the contents of a InterfaceNATStaticRulesRouterPolicy:

<table>
<thead>
<tr>
<th>Element</th>
<th>Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>staticRuleType</td>
<td>Enumeration</td>
<td>The type of local address to be translated by this static rule:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- “Static Host” – A single host requiring static address translation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- “Static Network” – A subnet requiring static address translation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- “Static Port” – A single port requiring static address translation.</td>
<td></td>
</tr>
<tr>
<td>original</td>
<td>Complex Type</td>
<td>A complex type element that identifies an IP address, or a network/host object representing the address(es) to be translated.</td>
<td></td>
</tr>
<tr>
<td>original.ipv4Data</td>
<td>String</td>
<td>A literal IP address.</td>
<td></td>
</tr>
<tr>
<td>original.networkObjectGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
<td></td>
</tr>
<tr>
<td>translated</td>
<td>Complex Type</td>
<td>A complex type element containing the addresses to which the Original Addresses are translated. This can contain either a specific IP address/network object or can be used to specify an interface. If the interface is specified, the IP addresses assigned to the interface is used as a translated address.</td>
<td></td>
</tr>
<tr>
<td>translated.originalIP</td>
<td>Complex Type</td>
<td>Complex Type that specifies an IP data or network policy object.</td>
<td></td>
</tr>
<tr>
<td>translated.originalIP.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
<td></td>
</tr>
<tr>
<td>translated.originalIP.networkObjectGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
<td></td>
</tr>
<tr>
<td>translated.interfaceGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a InterfaceRole Policy Object.</td>
<td></td>
</tr>
<tr>
<td>portRedirection</td>
<td>Complex Type</td>
<td>A complex type that specifies port information for the address translations. These parameters are available only when Static Port is the chosen rule type.</td>
<td></td>
</tr>
<tr>
<td>portRedirection.protocol</td>
<td>String</td>
<td>The communications protocol used for these ports: TCP or UDP.</td>
<td></td>
</tr>
<tr>
<td>Element. Sub Element</td>
<td>Type</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>portRedirection.localPort</td>
<td>unsignedInt</td>
<td>The port number on the source network. Valid values range from 1 to 65535.</td>
<td></td>
</tr>
<tr>
<td>portRedirection.globalPort</td>
<td>unsignedInt</td>
<td>The port number on the destination network that the router is to use for this translation. Valid values range from 1 to 65535.</td>
<td></td>
</tr>
<tr>
<td>settings</td>
<td>Complex Type</td>
<td>An optional complex type element that contains advanced options.</td>
<td></td>
</tr>
<tr>
<td>settings.noAlias</td>
<td>boolean</td>
<td>Disable automatic aliasing for the global IP address translation if true</td>
<td></td>
</tr>
<tr>
<td>settings.noPayload</td>
<td>boolean</td>
<td>Prohibit an embedded address or port in the payload from being translated if true.</td>
<td></td>
</tr>
<tr>
<td>settings.createExtTransEntry</td>
<td>boolean</td>
<td>Extended translation entries (addresses and ports) are created in the translation table if true.</td>
<td></td>
</tr>
</tbody>
</table>

Table 38: InterfaceNATStaticRulesRouterPolicy Class Definition
Figure 40: InterfaceNATStaticRulesRouterPolicy XML Definition
### 3.1.5.8 InterfaceNATDynamicRulesRouterPolicy


The InterfaceNATDynamicRulesRouterPolicy can reference an ACL PolicyObject and an InterfaceRole PolicyObject. The base orderId attribute defines the ordering of the dynamic rules.

The following table defines the contents of an InterfaceNATDynamicRulesRouterPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>trafficFlowAclObjectGID</td>
<td>ObjectIdentifier</td>
<td>References an access control list (ACL) Policy object GID whose entries define the addresses requiring dynamic translation.</td>
</tr>
<tr>
<td>translatedAddress</td>
<td>Complex Type</td>
<td>A complex type element that specifies the method and address(es) used for dynamic translation. Contains either an interface role object or address pool. If the interface role policy object is referenced then the globally registered IP address assigned to a particular interface will be used as the translated address.</td>
</tr>
<tr>
<td>translatedAddress.interfaceGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references an InterfaceRole Policy Object.</td>
</tr>
<tr>
<td>translatedAddress.address Pool</td>
<td>String</td>
<td>One or more address ranges, including the prefix, using the format min1-max1/prefix (in CIDR notation), where “prefix” represents a valid netmask. For example, 172.16.0.0–172.31.0.223/12.</td>
</tr>
<tr>
<td>settings</td>
<td>Complex Type</td>
<td>A complex type element that includes optional settings.</td>
</tr>
<tr>
<td>settings.enablePortTrans</td>
<td>Boolean</td>
<td>If true, the router uses port addressing (PAT) if supply of global addresses in the address pool is depleted; when false, PAT is not used.</td>
</tr>
<tr>
<td>settings.noTransVPN</td>
<td>Boolean</td>
<td>If true, address translation is not performed on VPN traffic. When false, the router performs address translation on VPN traffic in cases of overlapping addresses between the NAT ACL and the crypto ACL.</td>
</tr>
</tbody>
</table>

**Table 39: InterfaceNATDynamicRulesRouterPolicy Class Definition**
Figure 41: InterfaceNATDynamicRulesRouterPolicy XML Schema
3.1.5.9 **DeviceNATTimeoutsRouterPolicy**

A DeviceNATTimeoutsRouterPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a DeviceNATTimeoutsRouterPolicy specifies NAT timeout values for port address (overload) translations.

The following table defines the contents of a DeviceNATTimeoutsRouterPolicy:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxEntries</td>
<td>unsignedLong</td>
<td>The maximum number of entries allowed in the dynamic NAT table. Corresponds to a value between 1 and 2147483647. If not specified, it means that the number of entries in the table is unlimited.</td>
</tr>
<tr>
<td>timeoutInSecs</td>
<td>unsignedLong</td>
<td>The number of seconds after which dynamic translations expire; this does not apply to PAT (overload) translations. The default is 86400 seconds (24 hours).</td>
</tr>
<tr>
<td>udpTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to User Datagram Protocol (UDP) ports. The default is 300 seconds (5 minutes).</td>
</tr>
<tr>
<td>dnsTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to Domain Naming System (DNS) server connections. The default is 60 seconds.</td>
</tr>
<tr>
<td>tcpTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to Transmission Control Protocol (TCP) ports. The default is 86400 seconds (24 hours).</td>
</tr>
<tr>
<td>finRstTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied when a Finish (FIN) packet or Reset (RST) packet (both of which terminate connections) is found in the TCP stream. The default is 60 seconds.</td>
</tr>
<tr>
<td>icmpTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to Internet Control Message Protocol (ICMP) flows. The default is 60 seconds.</td>
</tr>
<tr>
<td>pptpTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to NAT Point-to-Point Tunneling Protocol (PPTP) flows. The default is 86400 seconds (24 hours).</td>
</tr>
<tr>
<td>synTimeoutInSecs</td>
<td>unsignedLong</td>
<td>The timeout value applied to TCP flows after a synchronous transmission (SYN) message (used for precise clocking) is encountered. The default is 60 seconds.</td>
</tr>
</tbody>
</table>

**Table 40: DeviceNATTimeoutsRouterPolicy Class Definition**
Figure 42: DeviceNATTimeoutsRouterPolicy
### 3.1.5.10 InterfaceNATAddressPoolFirewallPolicy

An InterfaceNATAddressPoolFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of an InterfaceNATAddressPoolFirewallPolicy manages the global address pools used in dynamic NAT rules. This policy is applicable for PIX, FWSM and pre-ASA 8.3.

The following table defines the contents of an InterfaceNATAddressPoolFirewallPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>interfaceGID</td>
<td>ObjectIdentifier</td>
<td>References a InterfaceRole Policy object GID interface on which the mapped IP addresses will be used.</td>
</tr>
<tr>
<td>poolId</td>
<td>unsignedInt</td>
<td>A unique identification number for this address pool, an integer between 1 and 2147483647. When configuring a dynamic NAT rule, the Pool ID is used to specify the pool of addresses to be used for translation.</td>
</tr>
<tr>
<td>ipAddressRange</td>
<td>Complex Type</td>
<td>A complex type element containing the address(es) to be assigned to this address pool. The address can contain a combination of literal IPv4 addresses and/or reference to network policy objects.</td>
</tr>
<tr>
<td>ipAddressRange.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>ipAddressRange.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>interfaceKeyword</td>
<td>String</td>
<td>If the “interface” keyword is specified it means port address translation is enabled on the specified interface.</td>
</tr>
</tbody>
</table>

#### Table 41: InterfaceNATAddressPoolFirewallPolicy Class Definition

```xml
<xs:complexType name="InterfaceNATAddressPoolFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="interfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="poolId" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
        <xs:element name="ipAddressRange" type="NetworkObjectsRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="interfaceKeyword" type="xs:string" fixed="interface" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### Figure 43: InterfaceNATAddressPoolFirewallPolicy XML Schema
3.1.5.11 DeviceNATTransOptionsFirewallPolicy

A DeviceNATTransOptionsFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a DeviceNATTransOptionsFirewallPolicy manages the options that affect network address translation for the selected security appliance. These settings apply to all interfaces on the device.

This policy is applicable for PIX, FWSM and ASA.

The following table defines the contents of a DeviceNATTransOptionsFirewallPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
</table>
| isEnableTrafficWithoutTrans| boolean   | It true, lets traffic pass through the security appliance without address translation. If this option is false, any traffic that does not match a translation rule will be dropped.  
Note: This option is available only on PIX 7.x, FWSM 3.x, and ASA devices. |
| isXlateByPass               | boolean   | If true, NAT session for imtranslated traffic are disabled.  
Note: This option is available only on FWSM 3.2 and higher |

Table 42: DeviceNATTransOptionsFirewallPolicy Class Definition

```xml
<xs:complexType name="DeviceNATTransOptionsFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="isEnableTrafficWithoutTrans" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
        <xs:element name="isXlateByPass" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 44: DeviceNATTransOptionsFirewallPolicy XML Schema
3.1.5.12  InterfaceNATTransExemptionsFirewallPolicy

An InterfaceNATTransExemptionsFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of an InterfaceNATTransExemptionsFirewallPolicy specifies rules that exempt traffic from address translation. Rules are evaluated sequentially in the order listed.

This policy is applicable for PIX, FWSM and pre-ASA 8.3.

The following table defines the contents of an InterfaceNATTransExemptionsFirewallPolicy:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRuleEnabled</td>
<td>boolean</td>
<td>If true, the rule is enabled and false indicates that the rule is disabled.</td>
</tr>
<tr>
<td>isExempt</td>
<td>boolean</td>
<td>If true, the rule identifies traffic that is exempt from NAT. If false, the rule identifies traffic that is not exempt from NAT.</td>
</tr>
<tr>
<td>realInterfaceGID</td>
<td>ObjectIdentifier</td>
<td>The device interface to which the rule is applied.</td>
</tr>
<tr>
<td>original</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the source hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network policy objects</td>
</tr>
<tr>
<td>original.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>original.interfaceRoleObjectGIDs</td>
<td>Object Identifier</td>
<td>List of Interface role Policy Object GIDs.</td>
</tr>
<tr>
<td>original.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>outsideNAT</td>
<td>boolean</td>
<td>True indicates rule outside keyword is defined on the NAT rule.</td>
</tr>
<tr>
<td>destinations</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the destination hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network policy objects</td>
</tr>
<tr>
<td>destinations.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>destinations.interfaceRoleObjectGIDs</td>
<td>Object Identifier</td>
<td>List of Interface role Policy Object GIDs.</td>
</tr>
<tr>
<td>destinations.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>fwsmAdvancedOptions</td>
<td>Complex Type</td>
<td>Advanced options applicable only for FWSM</td>
</tr>
<tr>
<td>fwsmAdvancedOptions.isTransDNSReplies</td>
<td>boolean</td>
<td>If true, the security appliance rewrites DNS replies so an outside client can resolve the name of an inside host using an inside DNS server, and vice versa.</td>
</tr>
<tr>
<td>fwsmAdvancedOptions.macTCPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of TCP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>fwsmAdvancedOptions.macUDPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of UDP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>Element. Sub Element</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fwsmAdvancedOptions.maxEmbConnections</td>
<td>UnsignedInt</td>
<td>The maximum number of embryonic connections allowed to form before the security appliance begins to deny these connections. Valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>fwsmAdvancedOptions.randomizeSeqNum</td>
<td>boolean</td>
<td>If true, the security appliance randomizes the sequence numbers of TCP packets</td>
</tr>
</tbody>
</table>

**Table 43: InterfaceNATTransExemptionsFirewallPolicy Class Definition**

```xml
<xs:complexType name="InterfaceNATTransExemptionsFirewallPolicy">
   <xs:complexContent>
      <xs:extension base="BasePolicy">
         <xs:sequence>
            <xs:element name="isRuleEnabled" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
            <xs:element name="isExempt" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
            <xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
            <xs:element name="original" type="NetworkObjectsRefs" minOccurs="1" maxOccurs="1"/>
            <xs:element name="outsideNAT" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
            <xs:element name="destinations" type="NetworkObjectsRefs" minOccurs="1" maxOccurs="1"/>
            <xs:element name="fwsmAdvancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

**Figure 45: InterfaceNATTransExemptionsFirewallPolicy XML Schema**
3.1.5.13 InterfaceNATDynamicRulesFirewallPolicy

An InterfaceNATDynamicRulesFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a InterfaceNATDynamicRulesFirewallPolicy specifies dynamic NAT and PAT rules. Rules are evaluated sequentially in the order listed.

This policy is applicable for PIX, FWSM and pre-ASA 8.3.

The following table defines the contents of an InterfaceNATDynamicRulesFirewallPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRuleEnabled</td>
<td>boolean</td>
<td>If true, the rule is enabled and false indicates that the rule is disabled.</td>
</tr>
<tr>
<td>realInterfaceGID</td>
<td>ObjectIdentifier</td>
<td>Maps to the device interface role policy object to which the rule applies.</td>
</tr>
<tr>
<td>poolId</td>
<td>Unsigned Int</td>
<td>The ID number of the pool of addresses used for translation. A value of zero to specify this as an identity NAT rule.</td>
</tr>
<tr>
<td>original</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the source hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network policy objects</td>
</tr>
<tr>
<td>original.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>original.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>outsideNAT</td>
<td>Boolean</td>
<td>If true, indicates the “outside” keyword is present on this NAT rule.</td>
</tr>
<tr>
<td>advancedOptions</td>
<td>Complex Type</td>
<td>Advanced options.</td>
</tr>
<tr>
<td>advancedOptions.isTranDNSReplies</td>
<td>boolean</td>
<td>If true, the security appliance rewrites DNS replies so an outside client can resolve the name of an inside host using an inside DNS server, and vice versa.</td>
</tr>
<tr>
<td>advancedOptions.maxTCPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of TCP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxUDPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of UDP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxEmbConnections</td>
<td>UnsignedInt</td>
<td>The maximum number of embryonic connections allowed to form before the security appliance begins to deny these connections. Valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.randomizeSeqNum</td>
<td>boolean</td>
<td>If true, the security appliance randomizes the sequence numbers of TCP packets</td>
</tr>
</tbody>
</table>

Table 44: InterfaceNATDynamicRulesFirewallPolicy Class Definition
Figure 46: InterfaceNATDynamicRulesFirewallPolicy XML Schema
### 3.1.5.14 InterfaceNATPolicyDynamicRulesFirewallPolicy

An InterfaceNATPolicyDynamicRulesFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a InterfaceNATPolicyDynamicRulesFirewallPolicy specifies dynamic translation rules based on source and destination addresses and services. Rules are evaluated sequentially in the order listed.

This policy is applicable for PIX, FWSM and pre-ASA 8.3.

The following table defines the contents of an InterfaceNATPolicyDynamicRulesFirewallPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRuleEnabled</td>
<td>boolean</td>
<td>If true, the rule is enabled and false indicates that the rule is disabled.</td>
</tr>
<tr>
<td>realInterfaceGID</td>
<td>ObjectIdentifier</td>
<td>Maps to the the device interface role policy object to which the rule applies.</td>
</tr>
<tr>
<td>poolId</td>
<td>Unsigned Int</td>
<td>The ID number of the pool of addresses used for translation. A value of zero to specify this as an identity NAT rule.</td>
</tr>
<tr>
<td>original</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the source hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network/interface role policy objects</td>
</tr>
<tr>
<td>original.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>original.interfaceRoleObjectId</td>
<td>ObjectIdentifierList</td>
<td>List of interface role policy objects.</td>
</tr>
<tr>
<td>original.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>outsideNAT</td>
<td>Boolean</td>
<td>If true, indicates the “outside” keyword is present on this NAT rule.</td>
</tr>
<tr>
<td>destinations</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the destination hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network/interface role policy objects</td>
</tr>
<tr>
<td>destinations.ipv4Data</td>
<td>String</td>
<td>A literal IP address.</td>
</tr>
<tr>
<td>destinations.interfaceRoleObjectId</td>
<td>ObjectIdentifierList</td>
<td>List of interface role policy objects.</td>
</tr>
<tr>
<td>destinations.networkObjectGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>services</td>
<td>Complex Type</td>
<td>Complex type containing services to which the rule applies. This can be a combination of service information in the format protocol/source-port/destination-port and/or references to Service Policy object</td>
</tr>
<tr>
<td>services.serviceData</td>
<td>String</td>
<td>The syntax for service specification is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>`{tcp</td>
</tr>
<tr>
<td>Element. Sub Element</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>**port_list_object} (destination_port_number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If only one port parameter, it refers tp the destination port (with a source port of “any”). For example, tcp/4443 means tcp, source port any, destination port 4443, while tcp/4443/Default Range means tcp, source port 4443, and destination port Default Range (generally 1-65535).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>services. serviceObjectGID</th>
<th>Object identifier</th>
<th>An ObjectIdentifier ID that references a Service Policy Object.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>advancedOptions</th>
<th>Complex Type</th>
<th>Advanced options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>advancedOptions.isTransDNSReplies</td>
<td>boolean</td>
<td>If true, the security appliance rewrites DNS replies so an outside client can resolve the name of an inside host using an inside DNS server, and vice versa.</td>
</tr>
<tr>
<td>advancedOptions.maxTCPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of TCP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxUDPConnPerRule</td>
<td>UnsignedInt</td>
<td>The maximum number of UDP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxEmbConnections</td>
<td>UnsignedInt</td>
<td>The maximum number of embryonic connections allowed to form before the security appliance begins to deny these connections. Valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.randomizeSeqNum</td>
<td>boolean</td>
<td>If true, the security appliance randomizes the sequence numbers of TCP packets</td>
</tr>
</tbody>
</table>

**Table 45: InterfaceNATPolicyDynamicRulesFirewallPolicy Class Definition**
Figure 47: InterfaceNATPolicyDynamicRulesFirewallPolicy XML Schema
### 3.1.5.15 InterfaceNATStaticRulesFirewallPolicy

An `InterfaceNATStaticRulesFirewallPolicy` extends from the base `BasePolicy` class and inherits all its attributes. An instance of an `InterfaceNATStaticRulesFirewallPolicy` specifies static translation rules for a security appliance. Rules are evaluated sequentially in the order listed.

This policy is applicable for PIX, FWSM and pre-ASA 8.3.

The following table defines the contents of an `InterfaceNATStaticRulesFirewallPolicy`:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRuleEnabled</td>
<td>boolean</td>
<td>If true, the rule is enabled and false indicates that the rule is disabled.</td>
</tr>
<tr>
<td>translationType</td>
<td>Enumeration</td>
<td>Type of translation for the rule - &quot;NAT&quot; or &quot;PAT&quot;.</td>
</tr>
<tr>
<td>realInterfaceGID</td>
<td>ObjectIdentifier</td>
<td>Maps to the device interface role policy object to which the rule applies.</td>
</tr>
<tr>
<td>mappedInterfaceGID</td>
<td>Object identifier</td>
<td>Maps to the interface role policy interface on which the translated addresses are to be used.</td>
</tr>
<tr>
<td>original</td>
<td>Complex Type</td>
<td>Complex type containing IP addresses for the source hosts and network objects to which the rule applies. Can contain multiple literal IP addresses and/or reference to network policy objects</td>
</tr>
<tr>
<td>original.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>original. networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>translated</td>
<td>Complex Type</td>
<td>A complex type element containing the translated addresses.</td>
</tr>
<tr>
<td>translated.ipv4Data</td>
<td>String</td>
<td>A literal IP address.</td>
</tr>
<tr>
<td>translated. networkObjectGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>translated.interfaceKeyword</td>
<td>String</td>
<td>A value of &quot;interface&quot; specifies that this keyword is present in the NAT rule</td>
</tr>
<tr>
<td>policyNAT</td>
<td>Complex Type</td>
<td>Complex Type containing Policy NAT details only if Policy NAT is enabled for this rule</td>
</tr>
<tr>
<td>policyNAT.destAddress</td>
<td>Complex Type</td>
<td>Complex Type containing the destination addresses.</td>
</tr>
<tr>
<td>policyNAT.destAddress.ipv4Data</td>
<td>String</td>
<td>A literal IP Address.</td>
</tr>
<tr>
<td>policyNAT.destAddress.networkObjectGIDs</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>policyNAT.services</td>
<td>Complex Type</td>
<td>Complex Type that specifies the services to which the rule applies. This can be a combination of service information in the format protocol/source-port/destination-port and/or references to Service Policy object</td>
</tr>
<tr>
<td>Element. Sub Element</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>policyNAT.services.service Data</td>
<td>String</td>
<td>The syntax for service specification is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{tcp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If only one port parameter, it refers to the destination port (with a source port of “any”). For example, tcp/4443 means tcp, source port any, destination port 4443, while tcp/4443/Default Range means tcp, source port 4443, and destination port Default Range (generally 1-65535).</td>
</tr>
<tr>
<td>policyNAT.services.serviceObjectGID</td>
<td>Object identifier</td>
<td>An ObjectIdentifier ID that references a Service Policy Object.</td>
</tr>
<tr>
<td>protocol</td>
<td>Enumeration</td>
<td>Protocol – “UDP”, “TCP”, “IP” to which the rule applies.</td>
</tr>
<tr>
<td>originalPort</td>
<td>Unsigned Int</td>
<td>If PAT is the selected Translation Type, this specifies the port number to be translated</td>
</tr>
<tr>
<td>translatedPort</td>
<td>Unsigned Int</td>
<td>If PAT is the selected Translation Type, this specifies the port number to which the original port number will be translated.</td>
</tr>
<tr>
<td>advancedOptions</td>
<td>Complex Type</td>
<td>Advanced options.</td>
</tr>
<tr>
<td>advancedOptions.isTransDNSReplies</td>
<td>boolean</td>
<td>If true, the security appliance rewrites DNS replies so an outside client can resolve the name of an inside host using an inside DNS server, and vice versa.</td>
</tr>
<tr>
<td>advancedOptions.maxTCPConnPerRule</td>
<td>Unsigned Int</td>
<td>The maximum number of TCP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxUDPConnPerRule</td>
<td>Unsigned Int</td>
<td>The maximum number of UDP connections allowed; valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.maxEmbyCnnections</td>
<td>Unsigned Int</td>
<td>The maximum number of embryonic connections allowed to form before the security appliance begins to deny these connections. Valid values are 0 through 65,535. If this value is set to zero, the number of connections is unlimited.</td>
</tr>
<tr>
<td>advancedOptions.randomizeSeqNum</td>
<td>boolean</td>
<td>If true, the security appliance randomizes the sequence numbers of TCP packets</td>
</tr>
</tbody>
</table>

Table 46: InterfaceNATStaticRulesFirewallPolicy Class Definition

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:complexType name="InterfaceNATStaticRulesFirewallPolicy">
    <xs:complexContent>
        <xs:extension base="BasePolicy">
            <xs:sequence>
                <xs:element name="isRuleEnabled" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
                <xs:element name="translationType" minOccurs="1" maxOccurs="1">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="NAT"/>
                            <xs:enumeration value="PAT"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
                <xs:element name="mappedInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
                <xs:element name="original" type="NetworkOrIPRef" minOccurs="1" maxOccurs="1"/>
                <xs:element name="translated" type="NetworkObjectRefs" minOccurs="1" maxOccurs="1"/>
                <xs:element name="policyNAT" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="destAddress" type="NetworkObjectsRefs" minOccurs="1" maxOccurs="1"/>
                            <xs:element name="services" minOccurs="1" maxOccurs="1">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="serviceData" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                                        <xs:element name="serviceObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="unbounded"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:complexType>
                            <xs:element name="protocol" type="IPTransportProtocol" minOccurs="1" maxOccurs="1"/>
                            <xs:element name="originalPort" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
                            <xs:element name="translatedPort" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
                            <xs:element name="advancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" maxOccurs="1"/>
                        </xs:complexType>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

Figure 48: InterfaceNATStaticRulesFirewallPolicy XML Schema
3.1.5.16 InterfaceNATManualFirewallPolicy

This policy is only applicable for device ASA version 8.3 or later. Starting ASA 8.3, the complete NAT representation is a combination of two key policy types. This consists of Manual NAT pre and post rules (section 1 and 3) and object NAT rules (section 2) as shown below.

This section covers the Manual NAT (InterfaceNATManualFirewallPolicy) and the following section covers the Object NAT (InterfaceNATObjectFirewallPolicy).

An InterfaceNATManualFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a InterfaceNATManualFirewallPolicy specifies the “manually” defined NAT rules on the device. The base orderId attribute that is inherited from the base policy specifies the ordering of these rules in the policy. The section element in the Policy data specifies whether this is a pre or post NAT rule.

The following table defines the contents of an InterfaceNATManualFirewallPolicy:

<table>
<thead>
<tr>
<th>Element</th>
<th>Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>isRuleEnabled</td>
<td>boolean</td>
<td>If true, the rule is enabled and false indicates that the rule is</td>
</tr>
</tbody>
</table>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>section</td>
<td>Enumeration</td>
<td>Specifies the rule section. Valid values are “1”, “2” and “3”. Following is the interpretation “1” → Indicates pre-NAT or NAT before rules “2” → Indicates Object NAT rules “3” → Indicates post-NAT or NAT rules after. For this policy type only “1” and “3” are possible. The BasePolicy orderId element will specify the “ordering” of the rules within this section.</td>
</tr>
<tr>
<td>realInterface</td>
<td>Complex Type</td>
<td>Reference to an interface role Policy Object or interface name</td>
</tr>
<tr>
<td>realInterface.realInterfaceGID</td>
<td>Object Identifier</td>
<td>Reference to an interface role Policy Object</td>
</tr>
<tr>
<td>realInterface.realInterfaceName</td>
<td>String</td>
<td>The real interface name</td>
</tr>
<tr>
<td>mappedInterface</td>
<td>Complex Type</td>
<td>Reference to an interface role Policy Object or name (Note: if both the real and mapped interfaces are not specified then the default to be assumed as “any”)</td>
</tr>
<tr>
<td>mappedInterface.mappedInterfaceGID</td>
<td>Object Identifier</td>
<td>Reference to an interface role Policy Object</td>
</tr>
<tr>
<td>mappedInterface.mappedInterfaceName</td>
<td>String</td>
<td>The mapped interface name.</td>
</tr>
<tr>
<td>source</td>
<td>Complex Type</td>
<td>Complex Type containing the Original and translated sources</td>
</tr>
<tr>
<td>source.natType</td>
<td>Enumeration</td>
<td>Specifies the type of translation rule either “Static” or “Dynamic”.</td>
</tr>
<tr>
<td>source.originalObjectGID</td>
<td>Object Identifier</td>
<td>Specifies the source address the NAT rule will translate. If this is a range or network, all addresses in the range or network are translated. This element includes an ObjectIdentifier ID that references a Network Policy Object.</td>
</tr>
<tr>
<td>source.translated</td>
<td>Complex Type</td>
<td>Complex Type containing either a pool of network address or reference to an interface object representation the translated source.</td>
</tr>
<tr>
<td>source.translated.ObjectGID</td>
<td>Object Identifier</td>
<td>Reference to a network Policy Object GID. Literal IPv4 addresses are not allowed for Manual NAT configurations.</td>
</tr>
<tr>
<td>source.translated.interfaceKeyword</td>
<td>String</td>
<td>The value of “interface” specifies that the interface keyword has been applied for the NAT rule.</td>
</tr>
<tr>
<td>source.translated.patPool</td>
<td>Complex Type</td>
<td>Complex Type containing the PAT options</td>
</tr>
<tr>
<td>source.translated.patPool.</td>
<td>Complex Type</td>
<td>Complex Type containing the PAT address pool</td>
</tr>
</tbody>
</table>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>patAddressPool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source.translated.patPool.</td>
<td>Object Identifier</td>
<td>Reference to a network policy object</td>
</tr>
<tr>
<td>patAddressPool.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>patPoolAddressGID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source.translated.patPool.</td>
<td>String</td>
<td>Interface to provide a Fallthrough Interface, it will be same as destination interface if the fixed string “interface” is used as the value for this element.</td>
</tr>
<tr>
<td>patAddressPool.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interfaceKeyword</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source.translated.patPool.</td>
<td>boolean</td>
<td>If true, it means a &quot;round robin&quot; cycling through available IP addresses and port numbers. This method assigns an address/port combination using each successive address in the pool; it then uses the first address again with a different port, proceeds to the second address again, and so on</td>
</tr>
<tr>
<td>isPatAllocatedInRoundRobin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td>Complex Type</td>
<td>Complex Type containing the the static translation of destination addresses</td>
</tr>
<tr>
<td>destination.natType</td>
<td>Enumeration</td>
<td>Specifies the type of translation rule either “Static” or “Dynamic”.</td>
</tr>
<tr>
<td>destination.originalObject</td>
<td>Complex Type</td>
<td>Reference to the original object</td>
</tr>
<tr>
<td>destination.</td>
<td>Object IdentifierList</td>
<td>Reference to a list of network Policy Object GIDs.</td>
</tr>
<tr>
<td>originalObject.networkObjectGIDs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination.</td>
<td>String</td>
<td>Destination IPV4 address.</td>
</tr>
<tr>
<td>originalObject.ipV4Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination.translatedObjectGID</td>
<td>ObjectIdentifier</td>
<td>Reference to the translated network object GID.</td>
</tr>
<tr>
<td>service</td>
<td>Complex Type</td>
<td>Complex Type that specifies port address translation.</td>
</tr>
<tr>
<td>service.originalObjectGID</td>
<td>Object Identifier</td>
<td>Reference to a Service Policy object that defines the service(s) to be translated.</td>
</tr>
<tr>
<td>service.transObjectGID</td>
<td>Object Identifier</td>
<td>Reference to a Service Policy object that defines the service(s) to be used for translation.</td>
</tr>
<tr>
<td>isTransDNSReplies</td>
<td>boolean</td>
<td>If true, addresses embedded in DNS replies that match this rule are rewritten.</td>
</tr>
<tr>
<td>Direction</td>
<td>Enumeration</td>
<td>Specifies whether a static NAT rule in a single direction only (“Unidirectional”); or dual rules (“Bidirectional”), one each for both directions (forward and reverse).</td>
</tr>
<tr>
<td>isNoProxyARP</td>
<td>boolean</td>
<td>If true, this disables proxy ARP on the specified Destination Interface. By default, all NAT rules include proxy ARP on the egress interface. A NAT Exempt rule is used to bypass NAT for both ingress and egress traffic, relying on route look-up to locate the egress interface. Thus, Proxy ARP should be disabled for NAT Exempt rules. (The NAT Exempt rules always take priority and...</td>
</tr>
<tr>
<td>Element. Sub Element</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>appear above all other NAT rules in the Translation Rules table.)</td>
</tr>
<tr>
<td>isRouteLookUp</td>
<td>boolean</td>
<td>If this option is true, the egress interface is determined using route look-up instead of using the specified Destination Interface. This must be true for a NAT Exempt rule.</td>
</tr>
</tbody>
</table>

**Table 47: InterfaceNATManualFirewallPolicy Class Definition**
Figure 49: InterfaceNATManualFirewallPolicy
3.1.5.17 InterfaceNAT64ManualFirewallPolicy

An InterfaceNAT64ManualFirewallPolicy represents a Unified (IPv6/IPv4) Manual NAT Rule. The Unified NAT Rules are supported from ASA 9.0 onwards. The network translations supported are

- IPv4 -> IPv6
- IPv6 -> IPv4
- IPv6 -> IPv6
- IPv4 -> IPv4


This policy is available starting from version 1.1 of the API.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isInterfaceIPv6</td>
<td>boolean</td>
<td>Use IPv6 Address of the interface</td>
</tr>
<tr>
<td>isNetToNet</td>
<td>boolean</td>
<td>Option for one-to-one mapping of single IPv6 address to an IPv4 server</td>
</tr>
</tbody>
</table>

Table 48: InterfaceNAT64ManualFirewallPolicy Class Definition

```xml
<xs:complexType name="InterfaceNAT64ManualFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="InterfaceNATManualFirewallPolicy">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="isInterfaceIPv6" type="boolean" minOccurs="0" maxOccurs="1"/>
        <xs:element name="isNetToNet" type="boolean" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 50: InterfaceNAT64ManualFirewallPolicy XML Schema
### 3.1.5.18 InterfaceNATObjectFirewallPolicy

An InterfaceNATObjectFirewallPolicy extends from the base BasePolicy class and inherits all its attributes. An instance of a InterfaceNATObjectFirewallPolicy specifies the object NAT rules on the device. The base order-id attribute that is inherited from the base policy specifies the ordering of these rules in the policy.

This policy is only applicable for device **ASA version 8.3 or later**

The following table defines the contents of an InterfaceNATObjectFirewallPolicy:

<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
</table>
| section              | Enumeration        | Specifies the rule section. Valid values are “1”, “2” and “3”. Following is the interpretation  
|                      |                    | “1” → Indicates pre-NAT or NAT before rules                               
|                      |                    | “2” → Indicates Object NAT rules                                         
|                      |                    | “3” → Indicates post-NAT or NAT rules after.                              
|                      |                    | For this policy type only “2” is allowed. The BasePolicy orderId element will specify the “ordering” of the rules within this section. |
| realInterface        | String             | Interface string                                                         |
| mappedInterface      | String             | Interface String                                                         |
| natType              | Enumeration        | Specifies the type of translation rule either “Static” or “Dynamic”.     |
| originalObjectGID    | String             | The source address the NAT rule will translate.                          |
| translated           | Complex Type       | Complex type that specifies whether the translation is based on either an address or interface. |
| translated.objectGID | Complex Type       | Complex Type containing the address definitions                          |
| translated.objectGID.ipv4Data | String   | A literal IP address.                                                   |
| translated.objectGID.networkObjectGID | Object identifier | An ObjectIdentifier ID that references a Network Policy Object.          |
| translated.objectGID.interfaceKeyword | String   | A value of “interface” indicates that the interface keyword is defined for this NAT rule. |
| translated.patPool   | Complex Type       | On ASA version 8.4.2 and later, a separate PAT Pool for a Dynamic NAT and PAT rule can be defined. The PAT Pool addresses are specified using this PAT Pool Address Translation field. This contains the PAT Pool options. |
| translated.patPool.patAddressPool | Complex Type | Containing the address information                                       |
| translated.patPool.patAddressPool.objectIdentifier | Object identifier | An ObjectIdentifier ID that references a Network Policy Object for PAT Pool. |

Cisco Security Manager 4.4 API Specification (Version 1.1)

Page 99
<table>
<thead>
<tr>
<th>Element. Sub Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>patPoolAddressGID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>translated.patPool.patAddressPool.interfaceKey word</td>
<td>String</td>
<td>A value of “interface” indicates that the interface keyword is defined for this NAT rule.</td>
</tr>
<tr>
<td>translated.patPool.patAddressPool.interfaceKey word</td>
<td>boolean</td>
<td>If this is true ASA Device uses Round Robin Allocation for PAT Pool</td>
</tr>
<tr>
<td>isTransDNSReplies</td>
<td>boolean</td>
<td>If true, addresses embedded in DNS replies that match this rule are rewritten.</td>
</tr>
<tr>
<td>isNoProxyARP</td>
<td>boolean</td>
<td>If true, do not proxy ARP on Destination Interface</td>
</tr>
<tr>
<td>isRouteLookUp</td>
<td>boolean</td>
<td>If true, perform Route Lookup for Destination Interface</td>
</tr>
<tr>
<td>service</td>
<td>Complex Type</td>
<td>Complex Type that defines configuration static port address translation. Application only for static rules</td>
</tr>
<tr>
<td>service.protocol</td>
<td>Enumeration</td>
<td>Protocol – “UDP”, “TCP” or “IP” to which the rule applies.</td>
</tr>
<tr>
<td>service.originalPort</td>
<td>Unsigned Int</td>
<td>The port on which the traffic enters the device.</td>
</tr>
<tr>
<td>service.transPort</td>
<td>Unsigned Int</td>
<td>The port number which is to replace the original port number.</td>
</tr>
</tbody>
</table>

**Table 49: InterfaceNATObjectFirewallPolicy Class Definition**
Figure 51: InterfaceNATObjectFirewallPolicy XML Schema
3.1.5.19 InterfaceNAT64ObjectFirewallPolicy

An InterfaceNAT64ObjectFirewallPolicy represents a Unified (IPv6/IPv4) Object NAT Rule. The Unified NAT Rules are supported from ASA 9.0 onwards. The network translations supported are:

- IPv4 -> IPv6
- IPv6 -> IPv4
- IPv6 -> IPv6
- IPv4 -> IPv4

InterfaceNAT64ObjectFirewallPolicy extends from the base InterfaceNATObjectFirewallPolicy class. An instance of a InterfaceNAT64ObjectFirewallPolicy denotes a single Unified Object NAT Rule.

This policy is available starting from version 1.1 of the API.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>isInterfaceIPv6</td>
<td>boolean</td>
<td>Use IPv6 Address of the interface</td>
</tr>
<tr>
<td>isNetToNet</td>
<td>boolean</td>
<td>Option for one-to-one mapping of single IPv6 address to an IPv4 server</td>
</tr>
</tbody>
</table>

Table 50: InterfaceNAT64ObjectFirewallPolicy Class Definition

```
<xs:complexType name="InterfaceNAT64ObjectFirewallPolicy">  
  <xs:complexContent>  
    <xs:extension base="InterfaceNATObjectFirewallPolicy">  
      <xs:sequence minOccurs="1" maxOccurs="1">  
        <xs:element name="isInterfaceIPv6" type="boolean" minOccurs="0" maxOccurs="1"/>  
        <xs:element name="isNetToNet" type="boolean" minOccurs="0" maxOccurs="1"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

Figure 52: InterfaceNAT64ObjectFirewallPolicy XML Schema
3.2 Methods
The configuration service defines the following methods:

1. GetServiceInfo
   a. Returns the service specific information including name of service, version, date…etc.
   b. This method should be called after authentication has returned a valid session cookie and prior to any other method to ensure the client is compatible with the service version currently running.

2. GetGroupList
   a. Returns the list of device groups defined in the system

3. GetDeviceListByCapability
   a. Returns the list of all devices that match one or more of the capabilities or wildcard in the system.

4. GetDeviceListByGroup
   a. Returns the list of all devices within a named group in the system.

5. GetDeviceConfigByGID
   a. Returns the device configuration identified by the object identifier specified in the method arguments.

6. GetDeviceConfigByName
   a. Returns the device configuration identified by the device’s name specified in the method arguments.

7. GetPolicyConfigByName
   a. Returns a policy configuration that is assigned to more than one device that has been assigned a name.

8. GetPolicyConfigByDeviceGID
   a. Returns the policy configuration belonging to the device’s object identifier that matches the particular type specified in the method arguments

9. GetPolicyListByDeviceGID
   a. Returns the list of policy names and their types belonging to the device’s object identifier specified in the method arguments

10. GetSharedPolicyListByType
    a. Returns a list of all the shared Policies defined in CSM for a given policy type.
3.2.1 Method GetServiceInfo

The GetServiceInfo method returns the service description, version information and pertinent attributes related to the service.

3.2.1.1 Request

An example of the method GetServiceInfo request is shown in the figure below. The fields in these messages are described in the table below.

URL:
https://hostname/nbi/configservice/GetServiceInfo

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<getServiceInfoRequest>
    <protVersion>1.0</protVersion>
    <reqId>123</reqId>
</getServiceInfoRequest>
```

Figure 53: Method GetServiceInfo Request Example

<table>
<thead>
<tr>
<th>HTTP/XML Content</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>getServiceInfoRequest</td>
<td>Object that contains request arguments</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>PUT</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

Figure 54: GetServiceRequest XML Schema
### 3.2.1.2 Response

An example of the GetServiceInfo response is shown in the figure below. The fields in these messages are described in the table below.

#### Figure 55: GetServiceInfo Response Example

#### Table 52: GetServiceInfo Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>XML Element &amp; Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>getServiceInfoResponse</td>
<td>Returns the service information of the configuration service</td>
</tr>
<tr>
<td>serviceVersion</td>
<td>The service version of the configuration service running</td>
</tr>
<tr>
<td>serviceName</td>
<td>The service name</td>
</tr>
<tr>
<td>serviceDesc</td>
<td>The service description that provides a more detailed explanation of the service capabilities and features</td>
</tr>
</tbody>
</table>

#### Figure 56: GetServiceInfoResponse XML Schema
3.2.2 Method GetGroupList
The GetGroupList method returns the list of devices matching a particular type or all devices if the wildcard argument is chosen.

3.2.2.1 Request
An example of the method GetGroupList request is shown in the figure below. The fields in these messages are described in the table below.

Figure 57: Method GetGroupList Request Example

Table 53: Method GetGroupList Request URL Argument Descriptions

<table>
<thead>
<tr>
<th>HTTP/XML Content</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupListRequest</td>
<td>Group list request that contains request arguments</td>
</tr>
<tr>
<td><strong>Element:</strong> includeEmptyGroups</td>
<td>An element that identifies whether the response should include groups that have no devices within them or not.</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>
3.2.2.2 Response

An example of the GroupListRequest response is shown in the figure below. The fields in these messages are described in the table below. In a GetGroupListResponse, the full config of the device will not be set. This is only set when the GetDeviceConfigByName or GetDeviceConfigByGID methods are called.
This XML example demonstrates a GetGroupList response:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<groupListResponse>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <deviceGroup>
    <gid>00000000-0000-0000-0000-00000000001</gid>
    <name>San Jose</name>
    <lastUpdateTime>2011-05-22T07:29:12.9Z</lastUpdateTime>
    <path>Vms\VirtualRoot\San Jose</path>
  </deviceGroup>
  <deviceGroup>
    <gid>00000000-0000-0000-0000-00000000002</gid>
    <name>Building 13</name>
    <lastUpdateTime>2011-05-22T07:29:12.9Z</lastUpdateTime>
    <path>Vms\VirtualRoot\San Jose\Building 13</path>
  </deviceGroup>
  <device>
    <gid>00000000-0000-0000-0000-00000000003</gid>
    <name>10.77.208.138</name>
    <lastUpdateTime>2011-05-26T00:11:53Z</lastUpdateTime>
    <osType>asa</osType>
    <osVersion>8.4(1)</osVersion>
    <imageName>disk0:/asa831-k8.bin</imageName>
    <sysObjectID>1.3.6.1.4.1.9.1.670</sysObjectID>
    <fullConfig/>
    <mgmtInterface>
      <type>Management</type>
      <identifier>mgmt</identifier>
      <ipInterface>
        <ipAddress>10.77.208.138/255.255.255.0</ipAddress>
      </ipInterface>
    </mgmtInterface>
    <interfaceList>
      <interface>
        <type>GigabitEthernet</type>
        <identifier>outside</identifier>
        <ipInterface>
          <ipAddress>20.10.30.42/255.255.255.0</ipAddress>
        </ipInterface>
      </interface>
    </interfaceList>
    <configState>committed</configState>
  </device>
</groupListResponse>
```

**Figure 59: GetGroupList Response Example**

**Table 54: GetGroupList Response Elements and Attributes Description**

<table>
<thead>
<tr>
<th>XML Element &amp; Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupListResponse</td>
<td>Returns a list of 0 or more device groups</td>
</tr>
<tr>
<td>XML Element &amp; Attributes</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Element List: DeviceGroup</td>
<td>A list of group elements</td>
</tr>
</tbody>
</table>

```
<xs:element name="groupListResponse" type="GroupListResponse"/>
<xs:complexType name="GroupListResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="deviceGroup" type="DeviceGroup"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 60: GetGroupList Response XML Schema**

Method specific errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>This error will be returned if the API is unable to retrieve the list of authorized devices for the user who is logged in.</td>
</tr>
<tr>
<td>2001</td>
<td>This error will be returned if API is not able to get the available groups from CSM Server for further processing.</td>
</tr>
<tr>
<td>2002</td>
<td>This error will be returned if API is not able to retrieve devices from CSM Server.</td>
</tr>
<tr>
<td>2003</td>
<td>This error will be returned if API is not able to find the config state of the device.</td>
</tr>
<tr>
<td>2004</td>
<td>This error will be returned if API is not able to retrieve interface details for a device.</td>
</tr>
<tr>
<td>2005, 2006</td>
<td>These errors will be returned if the API encounters an internal error when processing device specific data.</td>
</tr>
</tbody>
</table>

**Table 55: GetGroupList Method Error Codes**
3.2.3 Method GetDeviceListByCapability

The GetDeviceListByCapability method returns the list of devices matching one or more categories or all devices if the wildcard argument is chosen.

3.2.3.1 Request

An example of the method GetDeviceListByType request is shown in the figure below. The fields in these messages are described in the table below.

```
Figure 61: Method GetDeviceListByCapability Request Example

Table 56: Method GetDeviceListByCapability Request URL Argument Descriptions

<table>
<thead>
<tr>
<th>HTTP/XML Content</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDeviceListByCapabilityRequest</td>
<td>Device list request that contains request arguments</td>
</tr>
<tr>
<td><strong>Element</strong>: deviceCapability</td>
<td>One or more instances of an element that identifies the capability of the device being requested. Allowed values include:</td>
</tr>
<tr>
<td></td>
<td>- <strong>firewall</strong>: To return all ASA, PIX and FWSM devices.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ids</strong>: To return all IPS Devices</td>
</tr>
<tr>
<td></td>
<td>- <strong>router</strong>: To return routers</td>
</tr>
<tr>
<td></td>
<td>- <strong>switch</strong>: To return switches</td>
</tr>
<tr>
<td>Wildcard may be specified as</td>
<td><code>&lt;csm:deviceCapability&gt;*&lt;/csm:deviceCapability&gt;</code></td>
</tr>
</tbody>
</table>

HTTP Method: POST

HTTP Header: *asCookie*

The cookie returned by the login method that identifies the authentication session

Returns

200 OK + XML

401 Unauthorized

Cisco Security Manager 4.4 API Specification (Version 1.1)
3.2.3.2 Response

An example of the GetDeviceListByCapability response is shown in the figure below. The fields in these messages are described in the table below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<deviceListResponse>
    <protVersion>1.0</protVersion>
    <reqId>123</reqId>
    <deviceId>
        <gid>00000000-0000-0000-0000-12211312321</gid>
        <deviceCapability>firewall</deviceCapability>
        <ipv4Address>12.1.1.1</ipv4Address>
    </deviceId>
</deviceListResponse>
```

**Figure 63: GetDeviceListByCapability Response Example**

**Table 57: GetDeviceListByCapability Response Elements and Attributes Description**

<table>
<thead>
<tr>
<th>XML Element &amp; Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceListResponse</td>
<td>Returns a list of 0 or more devices that match the filter parameter passed in the method</td>
</tr>
<tr>
<td>Element List: DeviceId</td>
<td>A list of device ID elements</td>
</tr>
<tr>
<td>Attribute: Gid</td>
<td>The gid attribute of the device</td>
</tr>
<tr>
<td>Element: deviceCapability</td>
<td>One or more capabilities of the device (mandatory)</td>
</tr>
<tr>
<td>Element: deviceName</td>
<td>Name of the device (mandatory)</td>
</tr>
<tr>
<td>Element: ipv4Address</td>
<td>The IPv4 address of the device (optional)</td>
</tr>
</tbody>
</table>
Figure 64: DeviceListResponse XML Schema

The error codes documented for the GetGroupList method are also applicable for this method.
3.2.4 Method GetDeviceListByGroup

The GetDeviceListByGroup method returns the list of devices contained within a particular group or all devices if the wildcard argument is chosen. The group name path will be prepared by combining the entire path Items mentioned in the request body. The list of devices will be from the combination of the path Items. For example, the path Item is provided as San Jose alone and the CSM Server has sub groups under San Jose then the API will return all the devices matching group San Jose and other sub groups under it as well.

For example if the path items are San Jose and Building 14 then the devices matching group path “/San Jose/Building 14” and its sub group (if any) will be returned.

3.2.4.1 Request

An example of the method GetDeviceListByGroup request is shown in the figure below. The fields in these messages are described in the table below.

```
URL:
https://hostname/nbi/configservice/getDeviceListByGroup

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:
<?xml version="1.0" encoding="UTF-8"?>
<deviceListByGroupRequest>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <deviceGroupPath>
    <pathItem>building10</pathItem>
    <pathItem>4th floor</pathItem>
  </deviceGroupPath>
</deviceListByGroupRequest>
```

**Figure 65: Method GetDeviceListByGroup Request Example**

**Table 58: Method GetDeviceListByGroup Request URL Argument Descriptions**

<table>
<thead>
<tr>
<th>HTTP/XML Content</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDeviceListByGroupRequest</td>
<td>Device list request that returns a list of devices contained in the group</td>
</tr>
<tr>
<td>HTTP/XML Content</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Element: deviceGroupPath</strong></td>
<td>An element that identifies the device group being requested. Contains 1 or more <code>pathItem</code> entries. All the pathItems combined (separated by a slash “/”) will be treated as one group name. The path information from the response of the GetGroupList method can be used here. Each group in the GetGroupList has path attribute which provide the complete path name. Each element in the path (separated by a “/”) is encoded as pathItems. For example if device details for group includes a path such as “/VmsVirtualRoot/San Jose/Building 13” (obtained from the response of getGroupList API), then pathItems will be “VmsVirtualRoot”, “San Jose” and “Building 13”. The “VmsVirtualRoot” is a virtual “root node” for all groups.</td>
</tr>
<tr>
<td><strong>HTTP Method</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>HTTP Header: asCookie</strong></td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

```xml
<xs:element name="deviceListByGroupRequest" type="DeviceListByGroupRequest"/>
<xs:complexType name="DeviceListByGroupRequest">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="deviceGroupPath" type="DeviceGroupPath"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

*Figure 66: DeviceListByGroupRequest XML Schema*

### 3.2.4.2 Response

The GetDeviceListByGroup response and error codes are the same as method GetDeviceListByCapability response.
### 3.2.5 Method GetDeviceConfigByGID

The GetDeviceConfigByGID method returns a specific device object and its associated configuration based on the device id passed into the method. The user requesting the device configuration must have both the “view_device” and “view_cli” privilege on the device for which the config is being requested. The data in the “fullConfig” element that this method returns is equal to the data from a typical “show running-config” command executed on the device. This full configuration is returned from CSM’s Configuration Archive (CA) component. The CA is updated during device discovery and/or deployments. **NOTE:** This fullConfig will not contain any Out Of Band (OOB) configuration updates done directly on the device without using CSM.

Please also see Section 4 and Section 5.

### 3.2.5.1 Request

An example of the method GetDeviceConfigByGID request is shown in the figure below. The fields in these messages are described in the table below.

![Figure 67: Method GetDeviceConfigByGID Request Example](image)

<table>
<thead>
<tr>
<th>URL Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceConfigByGIDRequest</td>
<td>A global ID request</td>
</tr>
<tr>
<td>gid</td>
<td>The device global object identifier requested to be returned. <strong>NOTE:</strong> The device GID may be obtained by using any of the APIs - <code>getDeviceListByCapability</code> or <code>getDeviceListByGroup</code> or <code>GetGroupList</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>URL Attribute Name</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

**Figure 68: DeviceConfigByGIDRequest XML Schema**

3.2.5.2 Response
An example of the GetDeviceConfigByGID response is shown in the figure below. The fields in these messages are described in the table below.
Figure 69: GetDeviceConfigByGID Response Example

Table 60: GetDeviceConfigById Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>Element.Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceConfigResponse</td>
<td>Returns the device configuration that matches the object identifier</td>
</tr>
<tr>
<td>device</td>
<td>The device class as described in 2.1.3.</td>
</tr>
</tbody>
</table>
Method specific errors (in addition, the error codes defined for the GetGroupList method are also applicable here):

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>This error will be returned if the requested Device GID does not exist in the CSM Server.</td>
</tr>
<tr>
<td>2008</td>
<td>This error will be returned if API is not able to communicate with the configuration archive module.</td>
</tr>
<tr>
<td>2009</td>
<td>This error will be returned if API is not able to fetch the configuration from the Configuration Archive Module</td>
</tr>
<tr>
<td>2011</td>
<td>This error will be returned if the user is not authorized to view the configuration.</td>
</tr>
</tbody>
</table>

Table 61: GetDeviceConfigByGID Method Error Codes

3.2.6 Method GetDeviceConfigByName

The GetDeviceConfigByName method returns a specific device object and its associated configuration based on the device name passed into the method. The user requesting the device configuration must have both the “view_device” and “view_cli” privilege on the device for which the config is being requested. The data in the “fullConfig” element that this method returns is equal to the data from a typical “show running-config” command executed on the device. This full configuration is returned from CSM’s Configuration Archive (CA) component. The CA is updated during device discovery and/or deployments. **NOTE:** This fullConfig will not contain any Out Of Band (OOB) configuration updates done directly on the device without using CSM.

Please also see Section 4 and Section 5.

3.2.6.1 Request

An example of the method GetDeviceConfigByName request is shown in the figure below.. The fields in these messages are described in the table below.

---

**Figure 70: DeviceConfigResponse XML Schema**

```xml
<xs:element name="deviceConfigResponse" type="DeviceConfigResponse"/>
<xs:complexType name="DeviceConfigResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="device" type="Device"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```
Figure 71: Method GetDeviceConfigByName Request Example

Table 62: Method GetDeviceConfigByName Request URL Attribute Descriptions

<table>
<thead>
<tr>
<th>URL Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceConfigByNameRequest</td>
<td>A device name request</td>
</tr>
<tr>
<td>name</td>
<td>The device name requested to be returned</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

Figure 72: DeviceConfigByNameRequest XML Schema
3.2.6.2 Response

The response is the same as GetDeviceConfigByGID method.

Method specific errors (in addition, the error codes defined for the GetGroupList and GetDeviceConfigByGID methods are also applicable here):

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>This error will be returned if the requested device name does not exist in the CSM Server.</td>
</tr>
<tr>
<td>2012</td>
<td>This error will be returned if device name is null or empty in the request.</td>
</tr>
</tbody>
</table>

Table 63: GetDeviceConfigByName Method Error Codes
3.2.7 Method GetPolicyListByDeviceGID
The GetPolicyListByDeviceGID method returns the list of policy names and their types, for a particular device GID. The returned list only includes the supported policy types in the current version of the API. A policy type may also not be returned if the corresponding policy is not configured. To use this API user needs to have a view_device RBAC privilege.

Following is the list of policies supported in this version 1.0. Exact value from the policy type column need to be used in the request of the API’s GetPolicyByName and GetPolicyByDeviceGID methods.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceAccessRuleFirewallPolicy</td>
<td>Used to configure ACLs.</td>
</tr>
<tr>
<td>DeviceAccessRuleUnifiedFirewallPolicy</td>
<td>Used to configure Unified ACLs.</td>
</tr>
<tr>
<td>DeviceBGPRouterPolicy</td>
<td>Routing protocol.</td>
</tr>
<tr>
<td>DeviceNATTimeoutsRouterPolicy</td>
<td>Used for configuring Router NAT.</td>
</tr>
<tr>
<td>DeviceStaticRoutingFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>DeviceStaticRoutingRouterPolicy</td>
<td>Used in configuring Router static route.</td>
</tr>
<tr>
<td>InterfaceNAT64ManualFirewallPolicy</td>
<td>Used for configuring NAT64 policy</td>
</tr>
<tr>
<td>InterfaceNATAddressPoolFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATDynamicRulesFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATDynamicRulesRouterPolicy</td>
<td>Used for configuring Router NAT.</td>
</tr>
<tr>
<td>InterfaceNATManualFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATOObjectFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATPolicyDynamicRulesFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATRouterPolicy</td>
<td>Used for configuring Router NAT.</td>
</tr>
<tr>
<td>InterfaceNATStaticRulesFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
<tr>
<td>InterfaceNATStaticRulesRouterPolicy</td>
<td>Used for configuring Router NAT.</td>
</tr>
<tr>
<td>InterfaceNATTransExemptionsFirewallPolicy</td>
<td>Used for configuring Firewall NAT.</td>
</tr>
</tbody>
</table>

3.2.7.1 Request
An example of the method GetPolicyListByDeviceGID request is shown in the figure below. The fields in these messages are described in the table below.
Figure 73: Method GetPolicyListByDeviceGID Request Example

Table 64: Method GetPolicyListByDeviceGID Request URL Argument Descriptions

<table>
<thead>
<tr>
<th>HTTP/XML Content</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPolicyListByDeviceGID</td>
<td>Policy list request that contains request arguments</td>
</tr>
<tr>
<td>Element: gid</td>
<td>An element that identifies the device the policies are configured on.</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

Figure 74: PolicyListByDeviceGIDRequest XML Schema
### 3.2.7.2 Response

An example of the GetPolicyListByDeviceGID response is shown in the figure below. The fields in these messages are described in the table below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<policyListDeviceResponse>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <policyList>
    <policyDesc>
      <name>--local--</name>
      <type>DeviceOSFPolicy</type>
    </policyDesc>
    <policyDesc>
      <name>MySharedPolicy</name>
      <type>DeviceAccessRuleFirewallPolicy</type>
    </policyDesc>
  </policyList>
</policyListDeviceResponse>
```

**Figure 75: GetPolicyListByDeviceGID Response Example**

<table>
<thead>
<tr>
<th>XML Element &amp; Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>policyListResponse</td>
<td>Returns a list of 0 or more policy descriptors that are configured on the identified device</td>
</tr>
<tr>
<td>Element List: policyList</td>
<td>A list of policy descriptor elements</td>
</tr>
<tr>
<td><strong>Element: policyDesc</strong></td>
<td>The policy descriptor that includes its name and type. The type is usually the same name as the object name defined for that policy in the schema. For example for Firewall rules, the type will be “DeviceAccessRuleFirewallPolicy”. Name of the policy, will be --local-- if the policy is local/private to this device. Alternately the name will contain the user configured policy name (other than --local--) if the current policy is a “shared” policy (say something like “SharedAccessRule”).</td>
</tr>
</tbody>
</table>

Shared policies are policies that are shared across multiple devices. A shared policy name is unique for a specific policy type across the whole system. For example there can only be only policy named as SharedAccessRule for say a DeviceAccessRuleFirewallPolicy. And this SharedAccessRule can be applied to multiple devices that support that policy.
Figure 76: PolicyListDeviceResponse XML Schema

Method specific errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>This error will be returned if the requested device is not be configured with any of the supported policies.</td>
</tr>
<tr>
<td>2013</td>
<td>This error will be returned if API is not able to fetch the internal policy list from the CSM server.</td>
</tr>
</tbody>
</table>

Table 66: GetPolicyListByDeviceGID Method Error Codes

3.2.8 Method GetPolicyConfigByName

The GetPolicyConfigByName method returns a specific policy object and its associated configuration based on the shared policy name passed into the method. Access to this API requires a view_policy privilege on the policy being queried. This method must only be used to fetch data for shared policies (i.e. policies that are not -- local --). The applicable shared policies for a device are available via the GetPolicyListByDeviceGID method.

3.2.8.1 Request

An example of the method GetPolicyConfigByName request is shown in the figure below. The fields in these messages are described in the table below.
**Figure 77: Method GetPolicyConfigByName Request Example**

**Table 67: Method GetPolicyConfigByName Request URL Attribute Descriptions**

<table>
<thead>
<tr>
<th>Method Details</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>policyConfigByNameRequest</td>
<td>A name request and same attributes as previously defined and returned by the GetPolicyListByDeviceGID call.</td>
</tr>
<tr>
<td>Element: policyType</td>
<td>The policy type same as the types returned by the GetPolicyListByDeviceGID call. The list of allowable types are documented in the GetPolicyListByDeviceGID method section.</td>
</tr>
</tbody>
</table>

**HTTP Method**

POST

**HTTP Header: asCookie**

The cookie returned by the login method that identifies the authentication session

**Returns**

200 OK + XML

401 Unauthorized

```xml
<xs:element name="policyConfigByNameRequest" type="PolicyConfigByNameRequest"/>
<xs:complexType name="PolicyConfigByNameRequest">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="policyType" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 78: PolicyConfigByName Request XML Schema**
3.2.8.2 Response

An example of the response to the GetPolicyConfigByName request is shown in the figure below. The fields in these messages are described in the table below. The response contains the policy and referenced policy object data contained in the policy. Note: The response of this method is paginated. Please see section 2.2.1.1 for further details.
Figure 79: GetPolicyConfigByName Response Example
<table>
<thead>
<tr>
<th>Element/Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>policyConfigResponse</td>
<td>Returns the policy configuration that matches the object identifier</td>
</tr>
<tr>
<td>Policy</td>
<td>The policy class as described in 3.1.1.</td>
</tr>
</tbody>
</table>

**Table 68: GetPolicyConfigByName Response Elements and Attributes Description**

```
<xs:element name="policyConfigResponse" type="PolicyConfigResponse"/>
<xs:complexType name="PolicyConfigResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="policy" type="BasePolicy"/>
        <xs:element name="policyObject" type="BasePolicyObject"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 80: PolicyConfigResponse XML Schema**

The following are error codes specific to this method. There could be additional generic errors that the method might return in cases of error.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Returned if an internal error occurred while fetching config.</td>
</tr>
<tr>
<td>19</td>
<td>Returned if required input parameters are missing in the request.</td>
</tr>
<tr>
<td>20</td>
<td>Returned if the requested Policy Type is not correct.</td>
</tr>
<tr>
<td>22</td>
<td>Returned if the user is not authorized to view this policy data.</td>
</tr>
<tr>
<td>23</td>
<td>Returned if no configuration data is available for requested input parameters.</td>
</tr>
<tr>
<td>24</td>
<td>Returned if an internal error encountered by the server (when processing a policy type that is not supported).</td>
</tr>
</tbody>
</table>

**Table 69: GetPolicyConfigByName Method Error Codes**
3.2.9 Method GetPolicyConfigByDeviceGID

The GetPolicyConfigByDeviceGID method returns a specific policy and its associated policy objects based on the device id and policy type passed into the method. Access to this API requires a `view_policy` privilege on the policy being queried and also a `view_device` privilege on the device for which this policy is being fetched.

This method returns the applicable policy on a device (*irrespective of whether it is local or shared*). Thus this is often the most convenient method to read supported policy configuration of a device.

3.2.9.1 Request

An example of the method GetPolicyConfigByDeviceGID request is shown in the figure below. The fields in these messages are described in the table below.

![Figure 81: Method GetPolicyConfigByDeviceGID Request Example](image)

<table>
<thead>
<tr>
<th>Table 70: Method GetPolicyConfigByDeviceGID Request URL Attribute Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method Details</strong></td>
</tr>
<tr>
<td><code>policyConfigByDeviceGIDRequest</code></td>
</tr>
<tr>
<td><strong>HTTP Method</strong></td>
</tr>
<tr>
<td><strong>HTTP Header: asCookie</strong></td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
3.2.9.2 Response
The response of this method including the schema and error codes are the same as the GetPolicyConfigByName method described previously.

3.2.10 Method GetSharedPolicyNamesByType
This method returns the list of all the Shared Policies present in the Policy View of CSM for a given policy type.

3.2.10.1 REST Request:

URL:
http://hostname/configservice/getSharedPolicyListByType

HTTP Header:
Set-Cookie: asCookie=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:
<?xml version="1.0" encoding="UTF-8"?>
<policyNamesByTypeRequest>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <policyType>DeviceAccessRuleFirewallPolicy</policyType>
</policyNamesByTypeRequest>

Figure 83: getSharedPolicyNamesByType Request Example

Figure 82: PolicyConfigByDeviceGIDRequest XML Schema
Table 71: Method getSharedPolicyNamesByType Request URL Attribute Descriptions

<table>
<thead>
<tr>
<th>Method Details</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>policyNamesByTypeRequest</td>
<td>A request to get the list of all the Shared Policies defined in CSM for a given policy Type.</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>HTTP Header: asCookie</td>
<td>The cookie returned by the login method that identifies the authentication session</td>
</tr>
<tr>
<td>Returns</td>
<td>200 OK + XML</td>
</tr>
<tr>
<td></td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>
3.2.10.2 Response

Object:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<policyNamesResponse>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <policyType>DeviceAccessRuleFirewallPolicy</policyType>
  <policy>
    <name>policy-1</name>
    <deviceAssignments>
      <device>
        <deviceGID>00000000-0000-0000-0000-004294967308</deviceGID>
        <deviceName>1.1.1.1</deviceName>
      </device>
    </deviceAssignments>
  </policy>
  <policy>
    <name>policy-2</name>
    <deviceAssignments>
      <device>
        <deviceGID>00000000-0000-0000-0000-004294967309</deviceGID>
        <deviceName>1.1.1.2</deviceName>
      </device>
    </deviceAssignments>
  </policy>
</policyNamesResponse>
```

Figure 84: GetSharedPolicyNamesByType Response Example

<table>
<thead>
<tr>
<th>Element.Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>policyNamesResponse</td>
<td>Returns the policy Names and the Device Assignments for the Policy Type passed in the Request.</td>
</tr>
<tr>
<td>policyNamesResponse.policyType</td>
<td>The Policy Type passed in the Request.</td>
</tr>
<tr>
<td>policyNamesResponse.policy</td>
<td>A policy defined in CSM for the Requested Policy Type.</td>
</tr>
<tr>
<td>policyNamesResponse.policy.name</td>
<td>Name of the Shared Policy.</td>
</tr>
<tr>
<td>policyNamesResponse.policy.deviceAssignments</td>
<td>List of devices the Shared Policy is assigned to.</td>
</tr>
<tr>
<td>policyNamesResponse.policy.deviceAssignments.deviceGID</td>
<td>Device GID of the Device the Shared Policy is assigned to.</td>
</tr>
<tr>
<td>Element.Attribute Name</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>policyNamesResponse.policy.deviceAssignments.deviceName</td>
<td>Device Name of the Device the Shared Policy is assigned to.</td>
</tr>
</tbody>
</table>

### Table 72: GetSharedPolicyNamesByType Response Elements and Attributes Description

```xml
<xs:element name="policyNamesResponse" type="PolicyNamesResponse"/>
<xs:complexType name="PolicyNamesResponse">
    <xs:complexContent>
        <xs:extension base="BaseReqResp">
            <xs:sequence minOccurs="1" maxOccurs="1">
                <xs:element name="policyType" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="policy" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence minOccurs="1" maxOccurs="unbounded">
                            <xs:element name="deviceName" type="xs:string"/>
                            <xs:element name="deviceAssignments" minOccurs="0" maxOccurs="1">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="device" type="xs:string"/>
                                        <xs:element name="deviceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
                                        <xs:element name="deviceName" type="xs:string"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

**Figure 85: PolicyNamesResponse XML Schema**
4 CSM Events Service API

The events service API provides a mechanism for the client to receive the event notifications from the CSM server based on operations performed at CSM.

The event notifications will cover configuration updates initiated by CSM on the managed devices. Out-of-band (OOB) configuration changes on the managed devices will only be supported for ASA (version 7.2 and above) if the Health and Performance Monitoring (HPM) feature in CSM is enabled.

The event notifications will also cover Device Status change events. For the Device Status change event notifications to work the Device should be managed and monitored by CSM and HPM respectively.

4.1 Methods

The events service defines the following methods:

1. GetServiceInfo
   a. Returns the service specific information including name of service, version, date…etc.

2. EventSubscription
   a. Allows a client to register for event notifications.

4.1.1 Method GetServiceInfo

The GetServiceInfo method returns the service description, version information and pertinent attributes related to the service. The request, response and object model are the same as described in section 3.2.1.

4.1.2 Method EventSubscription

The eventSubscription method allows a CSM client to subscribe for event notifications that are filtered based on the criteria specified by the CSM client in the request. The event subscription API only supports configuration change notification syslog events and Device Status Change notification syslog events.

4.1.2.1 Request

An example of the method eventSubscription request is shown in the figure below. The fields in these messages are described in Table 73.
Figure 86: eventSubscription for ConfigChange XML Example

The above request will be sent by the client to register for events. The API only supports event notifications using the syslog protocol (the format of the event data over syslog can be XML or syslog plainText which is described in the table and subsequent sections).

All event subscriptions are linked to the active logged in session. If the session (that initiated the subscription request) is logged out or expires, then all subscriptions corresponding to the session are deleted. It is thus important for the client to keep the session alive if it needs to receive event notifications.

The following events are supported:

1. Config Change Event
   a. The event will be sent whenever a configuration change has been made and deployed on the network device identified in the event (either via CSM or out-of-band).

2. Device Status Change Event:
   a. This event will be sent whenever CSM detects a change in the device connectivity status, either UP or DOWN.

Table 73: eventSubscription Request Elements and Attributes Descriptions

<table>
<thead>
<tr>
<th>Element/Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventSubRequest</td>
<td>eventSubscription request allows the client to register a events handler for one or more classes</td>
</tr>
<tr>
<td><strong>Element</strong>.<strong>Attribute</strong> Name</td>
<td><strong>Definition</strong></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| eventSubRequest.op            | The operation being performed which is one of the enumerated list \{ add, delete \}.  
The add operation registers a new subscription for the event handler.op  
The delete operation deletes the subscription and the CSM client will no longer receive any events for that subscription ID. |
| eventSubRequest.subscriptionId| The subscription identifier that uniquely identifies this subscription registered by the CSM client. The subscription identifier will be returned with each event notification matching the filter specified in the request. |
| eventSubRequest.eventFilterItem| Object EventFilterItem specifies a filter that identifies a type of events that the CSM client requests notification on. |
| eventSubRequest.eventFilterItem.filterEventCategory| In 1.0 this is defined as “syslog”. Future types will be supported post-1.0. |
| eventSubRequest.eventFilterItem.filterEventType| Specifies the event category the client is interested in. Multiple filterEvents can be specified to listen to multiple types of events. configChange and deviceStatus are supported. |
| eventSubRequest.eventFilterItem.filterEventFormat| The event format that is used in the notification message. This is an enumerated list of formats. The list is as follows \{ xml, plainText \} |
| eventSubRequest.SyslogServer  | Is included in the subscription if the client wishes to receive the events over the syslog protocol at a specific destination and port number. Events that are not of syslog format originally will be encapsulated. |
| eventSubRequest.syslogServer.port | The UDP port number that the syslog relay will forward the event to |
| eventSubRequest.syslogServer.destAddress | The IPv4 Address that will receive the forwarded syslog events |

**HTTP Method**  
POST

**HTTP Header:** **asCookie**  
The cookie returned by the login method that identifies the authentication session

**Returns**  
200 OK + XML  
401 Unauthorized
4.1.2.2 Response

An example of the eventSubscription response is shown in the figure below. The fields in these messages are described in the table below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<eventSubResponse>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <subscriptionId>12345</subscriptionId>
</eventSubResponse>
```

Figure 88: eventSubscription Response Example
Table 74: eventSubscription Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>Element Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventSubResponse</td>
<td>Returns whether the subscription was successful or not. If successful it contains the subscriptionId sent by the client.</td>
</tr>
</tbody>
</table>

```xml
<xs:element name="eventSubResponse" type="EventSubResponse"/>
<xs:complexType name="EventSubResponse">
  <xs:complexContent>
    <xs:extension base="BaseReqResp">
      <xs:sequence>
        <xs:element name="subscriptionId" type="xs:string" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 89: EventSubResponse XML Schema

Following are some caveats for the syslog notifications:

- The syslog notifications will only be sent for devices currently managed by CSM i.e. the device must be added to the device inventory and must be “managed” by CSM.
- CSM may optionally deploy changes to Auto Update Server (AUS) and Cisco Networking Services (CNS) Configuration Engine based on how the device is managed in CSM. The AUS/CNS schemes provide an intermediate “staging system” for the configuration updates before these actually get deployed on the real device. Please see [http://www.cisco.com/en/US/docs/security/security_management/cisco_security_manager/security_manager/4.4/user/guide/dpman.html#wp938164](http://www.cisco.com/en/US/docs/security/security_management/cisco_security_manager/security_manager/4.4/user/guide/dpman.html#wp938164) for additional details. In cases where the deployment happens to CNS or AUS successfully, the event notification scheme will send a successful config change update (It is thus possible that at the time of sending the event, the config change might not be active on the real end device).

- Following are specific points for Out of Band (OOB) notifications:
  - The out of band change detection is currently only enabled for ASA devices (greater than version 7.2.x). Also OOB detection of ASA devices is only enabled if these devices are monitored by the Health and Performance Monitoring (HPM) feature of CSM.
  - The OOB notifications will only start detecting for OOB changes after at least “one successful event subscription is done”. OOB events prior to this registration will be unknown and will not be monitored.
  - The HPM component monitors devices every 5 or 10 minutes based on whether a device is actively or non-actively monitored. Hence OOB events will only be created once the HPM monitoring cycle/poll is completed. I.e it is likely that the OOB event will not be immediately generated as soon as any OOB change is done on a device.
  - The event service detects and notifies all OOB configuration changes including changed CLI/config lines that may not managed by CSM. Detection also includes any CLI that may be changed and reverted back. I.e. the system detects any configuration modified event on the device.
  - As a corner case, an OOB event may not be generated if a deployment via CSM is done immediately succeeding an OOB change (before the HPM module can detect the change). Consider the following case:
- At time 10:00 AM user does a OOB change on ASA1 that is being monitored every 10 minutes (next monitoring schedule for 10:09 AM)
- At time 10:03 AM user does a CSM deployment that overwrites the changes on ASA1 and deployment completes successfully
- At time 10:09 AM, the HPM poll cycle will not recognize the OOB change as it is overwritten by the CSM deployment done at 10:03 AM.

- Following are the specific points for deviceStatus notifications.
  - Notifications will be sent for All the Alerts with description “Device Polling” in Health and Performance Monitoring (HPM) feature of CSM. All the notifications will be sent whenever the CSM server is restarted. Henceforth only Device Status change notifications will be sent.
  - If the Alert description is “Device Polling: Connection Timedout” then a DEVICE_DOWN notification will be sent.
  - If the Alert description is “Device Polling: Connected” then a DEVICE_UP notification will be sent.
  - The HPM component monitors devices every 5 or 10 minutes based on whether a device is actively or non-actively monitored. Hence device status events will only be created once the HPM monitoring cycle/poll is completed. i.e. It is likely that the device might have gone down and come up, a device status change notification will not be sent in this case as the HPM would not have detected it.

The following are error codes specific to this method. There could be additional generic errors that the method might return in cases of error

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001</td>
<td>This indicates a duplicate subscription request was made during the event subscription. If two subscriptions are made using the same subscription id (for the same user session) then this error is returned.</td>
</tr>
<tr>
<td>4002</td>
<td>This error indicates that the Syslog destination IP address specified was invalid. The IP address must be in the a.b.c.d format without any mask address i.e. something like 192.168.10.10</td>
</tr>
<tr>
<td>4003</td>
<td>This error is returned if the user tries to delete a subscription that does not exist.</td>
</tr>
<tr>
<td>4004</td>
<td>This error indicates that the syslog port specified was invalid. Valid ports range between 1-65535.</td>
</tr>
<tr>
<td>4005</td>
<td>This error indicates an invalid subscriber ID was specified. Subscriber ID’s containing only whitespaces are not allowed.</td>
</tr>
<tr>
<td>4006</td>
<td>This error indicates that either the “Syslog Server” element or the “Event Filter” element was not specified. Both these elements must be specified for an add request only.</td>
</tr>
<tr>
<td>4007</td>
<td>This error indicates that either the “Syslog Server” element or the “Event Filter” element was specified for a delete operation. Both these elements must not be specified for an delete request.</td>
</tr>
</tbody>
</table>

Table 75: EventSubscription Method Error Codes

The following two sections elaborate on the content of syslog notification based on whether an XML based on plain text format was chosen during subscription.
4.1.2.3 Syslog XML Event Notifications

If the client has registered for syslog event forwarding using the SyslogServer option all events will be forwarded to
the registered destination IP and port number. The content of the syslog message may be transmitted in XML or
PlainText depending on the registration parameter specified by the client. If XML format is chosen the event
message will follow the schema as defined below.

Following is an example of a configuration change notification:

```
<n:configChangeEvent xmlns:n="csm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="events.xsd">
  <eventType>syslog</eventType>
  <eventCategory>configChange</eventCategory>
  <time>1697-02-01T00:00:00Z</time>
  <content>this is a syslog event</content>
  <srcIP>12.1.1.1</srcIP>
  <srcGID>00000000-0000-0000-0000-000000000890</srcGID>
  <srcDns>cisco.com</srcDns>
  <srcOSType>asa</srcOSType>
  <updateType>NO_OOB</updateType>
</n:configChangeEvent>
```

Figure 90: Configuration Change Notification Example

The above event notification is sent by the server to the registered syslog listener and is based on the following
elements and attributes.

Table 76: ConfigChangeEvent Data Element Descriptions

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>configChangeEvent</td>
<td>Root element indicating the CSM specific configuration change event notification.</td>
</tr>
<tr>
<td>configChangeEvent.eventType</td>
<td>Indicates this message is a syslog message</td>
</tr>
<tr>
<td>configChangeEvent.eventCategory</td>
<td>Indicates the event category which is configChange</td>
</tr>
<tr>
<td>configChangeEvent.subscriptionId</td>
<td>The original subscription ID used by the client that is now &quot;echoed&quot; back.</td>
</tr>
<tr>
<td>configChangeEvent.time</td>
<td>The time this event was generated</td>
</tr>
<tr>
<td>configChangeEvent.content</td>
<td>Any additional message associated with the configuration change.</td>
</tr>
<tr>
<td>configChangeEvent.srcIP</td>
<td>The IP address of the device that was changed</td>
</tr>
<tr>
<td>configChangeEvent.srcGID</td>
<td>The GID of the device that was changed</td>
</tr>
<tr>
<td>configChangeEvent.srcDns</td>
<td>The DNS name of the device that was changed (if any).</td>
</tr>
<tr>
<td>configChangeEvent.srcOSType</td>
<td>Indicates the device type ASA, FWSM, IOS, PIX.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| configChangeEvent.deploymentType | Indicates the type of deployment *(Not applicable in case of an out of Band change where it is set to “Unknown”).* Allowable values are:  
  - Device ➞ Deployment to a device  
  - File ➞ Deployment to a file  
  - AUS ➞ Deployment to the Auto Update Server  
  - CNS ➞ Deployment to Cisco Networking Services, Configuration Engine  
  - TMS ➞ Deployment to a Token Management Server                                                                                                                                 |
| configChangeEvent.updateType  | NO_OOB ➞ Indicates that this configuration update is not an Out of Band (OOB) change i.e. the configuration update to the device was done via Cisco Security Manager.  
  OOB ➞ Indicates that this configuration update is an OOB change done “outside” of Cisco Security Manager. **CSM will only have the capability to detect OOB changes for ASA (version 7.2 and above) devices alone.**                                                                 |

The following figure shows the schema for those event notifications:
Figure 91: Event XML Schema

Cisco Security Manager 4.4 API Specification (Version 1.1)
Following is an example of a Device Status change notification:

```
<n:deviceStatusEvent xmlns:n="csm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="events.xsd">  
    <subscriptionId>100</subscriptionId>  
    <eventType>syslog</eventType>  
    <eventCategory>deviceStatus</eventCategory>  
    <time>2013-01-25T04:47:59.082Z</time>  
    <content>Connection Error</content>  
    <srcIP>12.1.1.1</srcIP>  
    <srcGID>00000000-0000-0000-0000-000000000890</srcGID>  
    <srcDns>cisco.com</srcDns>  
    <srcOSType>asa</srcOSType>  
    <updateType>DEVICE_DOWN</updateType>  
</n:deviceStatusEvent>
```

**Figure 92: Device Status Notification Example**

The above event notification is sent by the server to the registered syslog listener and is based on the following elements and attributes.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceStatusEvent</td>
<td>Root element indicating the Device Status event notification.</td>
</tr>
<tr>
<td>deviceStatusEvent.eventType</td>
<td>Indicates this message is a syslog message</td>
</tr>
<tr>
<td>deviceStatusEvent.eventCategory</td>
<td>Indicates the event category which is deviceStatus</td>
</tr>
<tr>
<td>deviceStatusEvent.subscriptionId</td>
<td>The original subscription ID used by the client that is now “echoed” back.</td>
</tr>
<tr>
<td>deviceStatusEvent.time</td>
<td>The time this event was generated.</td>
</tr>
<tr>
<td>deviceStatusEvent.content</td>
<td>Any additional message associated with the configuration change.</td>
</tr>
<tr>
<td>deviceStatusEvent.srcIP</td>
<td>The IP address of the device that was changed</td>
</tr>
<tr>
<td>deviceStatusEvent.srcGID</td>
<td>The GID of the device that was changed</td>
</tr>
<tr>
<td>deviceStatusEvent.srcDns</td>
<td>The DNS name of the device that was changed (if any).</td>
</tr>
<tr>
<td>deviceStatusEvent.srcOSType</td>
<td>Indicates the device type ASA, FWSM, IOS, PIX.</td>
</tr>
<tr>
<td>deviceStatusEvent.updateType</td>
<td>DEVICE_DOWN ➔ The Device is not reachable from the CSM Server.</td>
</tr>
<tr>
<td></td>
<td>DEVICE_UP ➔ The Device has come up and is reachable from the CSM Server.</td>
</tr>
</tbody>
</table>

The following figure shows the schema for those event notifications:
<xs:simpleType name="OSType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="ios"/>
    <xs:enumeration value="fwsrm"/>
    <xs:enumeration value="asa"/>
    <xs:enumeration value="ips"/>
    <xs:enumeration value="pix"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="DeviceSpecificEvent">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="deviceStatusEvent" type="DeviceStatusEvent" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceStatusEvent">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="updateType" type="UpdateType" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="UpdateType">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="deviceStatusEvent" type="DeviceStatusEvent" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeploymentType">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="event" type="Event"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Event">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="name"/>
        <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="BaseEventDetails">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="eventType" type="EventType" minOccurs="1" maxOccurs="1"/>
        <xs:element name="eventCategory" type="EventCategory" minOccurs="1" maxOccurs="1"/>
        <xs:element name="time" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
        <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeploymentType">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="deviceStatusEvent" type="DeviceStatusEvent" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Event">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="name"/>
        <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="BaseEventDetails">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="eventType" type="EventType" minOccurs="1" maxOccurs="1"/>
        <xs:element name="eventCategory" type="EventCategory" minOccurs="1" maxOccurs="1"/>
        <xs:element name="time" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
        <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceSpecificEvent">
  <xs:complexContent>
    <xs:restriction base="xs:complexType">
      <xs:sequence>
        <xs:element name="deviceStatusEvent" type="DeviceStatusEvent" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
4.1.2.4 Syslog PlainText Event Notifications

The notifications in the native format will be sent using the syslog protocol if the registered `filterEventFormat` is specified as `plainText` during event subscription.

The following example shows a configuration change notification:

```plaintext
admin_job_2011-08-29 08:30:12.148,10.104.52.71,00000000-0000-0000-0000-017179869189,NO_DOMAIN
NAME DEFINED,ios,Device,NO_OOB.
```

The following example shows a Device Status change notification:

```plaintext
[Thu Jan 24 20:47:59 PST 2013] syslog-deviceStatus-102:,10.104.52.71,00000000-0000-0000-0000-042949673287,default.domain.invalid,asa,DEVICE_DOWN
```

The standard format for this message is

```plaintext
[time-stamp]<eventType>-<eventCategory>-<subscriptionId>: {Comma separated list of event details –<content>,<srcIp>,<srcGID>,<srcDNS>,<srcOSType>,<deploymentType>,<updateType>} }
```

The order of elements in the comma separated list is in the order of elements in the specific event type defined in the XML schema. If the device does not have an IP address then the string “NO IP DEFINED” (srcIP) is contained in the message and “NO DOMAIN NAME DEFINED” is used if there is no source DNS (srcDNS).
5 CSM Utility Service API

The utility service provides general utilities for access to functions on the network devices. The Utility Service API’s always fetch the device configuration data directly from the device. The Configuration Service API (defined in section 3) on the other hand always fetches the data from the CSM database.

Faster responses and better efficiency is obtained using the Configuration Service API rather than the Utility Service API. This is because the Utility Service API communicates with the device during a method request and there is an additional communication overhead between the CSM application and the network device. This may additionally entail an increased load on the network device.

It is thus recommended to use the Utility Service API under the following circumstances:

- If the corresponding configuration data is not supported by the Configuration Service API
- If the corresponding configuration data is administratively not managed in CSM. By default all applicable policies are managed in CSM (see the **Policy Management** CSM Administration screen)
- If out-of-band changes are made on the end device (in such cases CSM might not have the updated configuration in its database)

5.1 Object Model

The following objects are defined in the utility service

- **DeviceReadOnlyCLICmds**
  - Identifies a device and a CLI command to be executed against that device
- **DeviceCmdResults**
  - Identifies a set of device command results
- **DeviceCmdResult**
  - Identifies a single device and the command results

```
<xs:simpleType name="Result">
  <xs:restriction base="xs:token">
    <xs:enumeration value="ok"/>
    <xs:enumeration value="timeout"/>
    <xs:enumeration value="failed"/>
  </xs:restriction>
</xs:simpleType>
```

*Figure 94: Result XML Schema*

5.2 Methods

The utility service defines the following methods:

1. GetServiceInfo
a. Returns the service specific information including name of service, version, date…etc.

2. execDeviceReadOnlyCLICmd
   a. Executes the identified CLI command on the device specified in the method argument.

### 5.2.1 Method GetServiceInfo

The GetServiceInfo method returns the service description, version information and pertinent attributes related to the service. The URL for this request is `https://hostname/nbi/utservice/GetServiceInfo`. The request, response and object model are the same as described in section 3.2.1.
5.2.2 Method execDeviceReadOnlyCLICmds

The execDeviceReadOnlyCLICmds method executes a set of commands against a list identified devices and returns the result of each command back to the CSM Client. This method will use the read-only credentials the CSM server has and therefore may not be used to execute any commands that require credentials beyond read-only for the device. This method is not implemented for IPS/IDS devices.

The set of commands that can be executed by this method is read-only commands such as statistics, monitoring commands that provide additional information about the operation of the particular device. For example, a client application may invoke CLI commands to gather statistics from the device via this command. A CSM view_cli and view_device privilege is necessary to use this method. This method fetches configuration information for one device at a time.

NOTE: If show command output has any W3C XML standard reserved symbols then the API response will encode any such special characters as per W3C standards. For example “show version” output from a multi context device can have output as <context>, then response will appear with named character reference as &lt; context&gt;. For more details refer http://www.w3.org/TR/REC-xml/#syntax

5.2.2.1 Request

An example of the method execDeviceReadOnlyCLICmds request is shown in the figure below. The fields in these messages are described in the table below.

URL:

https://hostname/nbi/utilservice/execDeviceReadOnlyCLICmds

HTTP Header:

Set-Cookie: asCookie=732423sdafs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=.hostdomain.com

XML Argument:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<execDeviceReadOnlyCLICmdsRequest>
  <protVersion>1.0</protVersion>
  <reqId>123</reqId>
  <deviceReadOnlyCLICmd>
    <deviceIP>12.1.1.1</deviceIP>
    <cmd>show</cmd>
    <argument>run all</argument>
    <execTimeout>5</execTimeout>
  </deviceReadOnlyCLICmd>
</execDeviceReadOnlyCLICmdsRequest>
```

Figure 95: Method execDeviceReadOnlyCLICmds Request Example
Table 78: Method execDeviceReadOnlyCLICmds Request Elements and Attributes Descriptions

<table>
<thead>
<tr>
<th>Element, Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>execDeviceReadOnlyCLICmdsRequest</td>
<td>The execDeviceReadOnlyCLICmds request executes a command on the set of identified devices</td>
</tr>
<tr>
<td>deviceReadOnlyCLICmd</td>
<td>The device command details to be executed. The choice of deviceIP; deviceName or deviceGID to identify the device that the command is executed against is given. deviceName and deviceGID will perform more efficiently than deviceIP as they avoid having to search for the device credentials.</td>
</tr>
<tr>
<td>deviceIP</td>
<td>The device IP address that the command will be executed against</td>
</tr>
<tr>
<td>deviceName</td>
<td>The device Name that the command will be executed against</td>
</tr>
<tr>
<td>deviceGID</td>
<td>The device object identifier that the command will be executed against</td>
</tr>
<tr>
<td>cmd</td>
<td>Fixed command “show”. The regex allows mixed case [sS][hH][oO][wW] argument</td>
</tr>
<tr>
<td>argument</td>
<td>The show command arguments. Like “run” to show the running config of the device or “access-list” to show the access list details.</td>
</tr>
<tr>
<td>execTimeout</td>
<td>The execTimeout attribute optionally takes timeout in seconds. The execTimeout attribute will take be an unsigned Integer. Default is set to 180 Seconds. CSM will try the exec command for 3 times with a delay of 15 seconds. Each trial will have a timeout of the value provided for the execTimeout attribute.</td>
</tr>
<tr>
<td>reqld</td>
<td>The request identifier that uniquely identifies the request/response transaction pair between the CSM client and CSM server</td>
</tr>
</tbody>
</table>

**HTTP Method**

POST

**HTTP Header**: asCookie

The cookie returned by the login method that identifies the authentication session

**Returns**

200 OK + XML

401 Unauthorized
Figure 96: ExecDeviceReadOnlyCLICmdsRequest XML Schema

5.2.2.2 Response
An example of the execDeviceReadOnlyCLICmds response is shown in the figure below. The fields in these messages are described in the table below.
<?xml version="1.0" encoding="UTF-8"?>
<execDeviceReadOnlyCLICmdsResponse>
<reqId>123</reqId>
<deviceCmdResult>
<deviceIP>12.1.1.1</deviceIP>
<deviceName>rtr.cisco.com</deviceName>
<deviceGID>00000000-0000-0000-0000-261993005068</deviceGID>
<result>ok</result>
<resultContent>
FWSM Firewall Version 3.1(16) &lt;context&gt; 
Device Manager Version 5.1(1)
Compiled on Wed 29-Jul-09 02:10 by fwsmblid
U27-FWSM up 5 days 3 hours
Hardware:   WS-SVC-FWM-1, 1024 MB RAM, CPU Pentium III 1000 MHZ
Flash STI Flash 8.0.0 @ 0xc321, 20MB
 0: Int: Not licensed : irq 5
 1: Int: Not licensed : irq 7
 2: Int: Not licensed : irq 11
Licensed features for this platform:
Maximum Interfaces : 100
Inside Hosts : Unlimited
Failover : Active/Active
VPN-DES : Enabled
VPN-3DES-AES : Enabled
Cut-through Proxy : Enabled
Guards : Enabled
URL Filtering : Enabled
Security Contexts : 250
GTP/GPRS : Disabled
VPN Peers : Unlimited
Serial Number: SAD11420A0N
Running Activation Key: 0xe3bbe7c 0x0b82b2ba 0x4014b998 0x6bef38ad
Configuration has not been modified since last system restart.
</resultContent>
</deviceCmdResult>
</execDeviceReadOnlyCLICmdsResponse>

Figure 97: execDeviceReadOnlyCLICmds Response Example

Table 79: execDeviceReadOnlyCLICmds Response Elements and Attributes Description

<table>
<thead>
<tr>
<th>Element.Attribute Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>execDeviceReadOnlyCLICmdsResponse</td>
<td>Returns command result</td>
</tr>
<tr>
<td>serviceVersion</td>
<td>The service version of the configuration service running</td>
</tr>
<tr>
<td>deviceCmdResult</td>
<td>The command result details for the specific device</td>
</tr>
<tr>
<td>deviceIP</td>
<td>The device IP address if available.</td>
</tr>
<tr>
<td>deviceName</td>
<td>The device Name of the device that returned the result</td>
</tr>
<tr>
<td>deviceGID</td>
<td>The device object identifier returned the result</td>
</tr>
<tr>
<td>result</td>
<td>The command result enumeration { success, generalFailure, timeout }</td>
</tr>
<tr>
<td>resultContent</td>
<td>The command result content is the result was successfully</td>
</tr>
</tbody>
</table>
The following are error codes specific to this method. There could be additional generic errors that the method might return in cases of error.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>This error will be returned if the API is unable to execute the show commands on the network device.</td>
</tr>
<tr>
<td>3001</td>
<td>This error will be returned if CSM Server does not have any device with the requested IP Address.</td>
</tr>
<tr>
<td>3002</td>
<td>This error will be returned if the requested device is an IPS Device.</td>
</tr>
<tr>
<td>3003</td>
<td>This error will be returned if API encounters any issues when contacting the device, for example send a request for a device with invalid credentials.</td>
</tr>
<tr>
<td>27</td>
<td>Executimeout Failure: No Response from the device within the timeout.</td>
</tr>
</tbody>
</table>

**Table 80: ExecDeviceReadOnlyCLICmdsRequest Method Error Codes**
6 API Scaling

The API will support a variety of deployments. The following guidelines should be followed.

1) There is no hard limit on the number of devices supported by the API. The current CSM recommendation is to have approximately 500 devices per server with event management and other features enabled.
7 CSM Client Protocol State Machine

7.1.1 Overview

There are two pre-requisites before a CSM Client can make use of the services on the CSM Server. The 1st prerequisite is that the client authenticates to the CSM Server. The 2nd prerequisite is that the client verifies that all necessary services are active on the CSM server and the versions of those services match the version expected on the client. This process will return an authentication cookie that the client must use in subsequent calls to the interface. This flow is shown in the figure below.

Figure 99: Client Session Initiation Flowchart

Once a session has been established the client may access the service methods for config, events and utility services.
A client that has registered for heartbeat callback should follow the flow defined below.

Figure 100: Client Heartbeat Processing Flowchart

A client that has not registered for heartbeat callback but wishes to maintain an active session on the CSM server must call the ping method periodically.

Figure 101: Client Ping Processing Flowchart
7.1.2 Using the configuration and event service

The CSM client may access the API to read configuration of the devices supported by the CSM server. The client may also register for change notifications whenever a configuration change occurs. The following flowchart shows the high-level processing on the client.

Following is a typical flow to process policy data in raw format:

a. Login (and setup a background thread or process to keep the session alive via a ping or heartbeat)
b. Use one of GetGroupList (or) GetDeviceListByCapability (or) GetDeviceListByGroup to get the list of devices in the system
c. For each of the devices in the list:
   i. Call GetDeviceConfigByGID (or) GetDeviceConfigByName to get the raw config data
d. Logout

Client would additionally need to subscribe to events (see Event Service API in this document), to make the client refresh the configuration if the configuration gets updated. The GetDeviceConfigByGID and
GetDeviceConfigByName only return data that is archived inside CSM database. Any updates to the device outside of CSM (out of band) is not available and can only be fetched using the Utility Service API.

Following is a typical flow to process policy data in a policy object model **structured format**:

a. Login (and setup a background thread or process to keep the session alive via a ping or heartbeat)
b. Use one of GetGroupList (or) GetDeviceListByCapability (or) GetDeviceListByGroup to get the list of devices in the system
c. For each of the devices in the list:
   i. Call GetPolicyListByDeviceGID to find out what “policy types” are configured/supported on a device
   ii. For each of these “Policy Types” call the GetPolicyConfigByDeviceGID or GetPolicyConfigByName (for named/shared policies)
d. Logout

Client would additionally need to subscribe to events (see Event Service API in this document), to make the client refresh the configuration if the configuration gets updated. The GetDeviceConfigByGID and GetDeviceConfigByName only return data that is archived inside CSM database. Any updates to the device outside of CSM (out of band) is not available and can only be fetched using the Utility Service API.
8 Sample API Client Programs

NOTE: All the sample programs in this section are provided for simple demonstration purposes only. The programs may need to be appropriately enhanced or modified for use in production systems.

Running the java sample programs mentioned below need the following to be setup:

1. The http-client and http-core jars from Apache-commons http://hc.apache.org must be included in the java CLASSPATH.
2. The commons-logging jar from Apache-commons http://commons.apache.org/logging/index.html must be included in the java CLASSPATH
3. Define a CLASSPATH environment variable in your command shell include all the required jars and other folders as necessary
4. A client.properties file must be defined and passed in as a parameter as a java program argument. The format of the file is as follows:

```
USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST=<xml version="1.0" encoding="UTF-8"?>
   <csm:pingRequest xmlns:csm="csm">
      <protVersion>1.0</protVersion>
      <reqId>3</reqId>
   </csm:pingRequest>
# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/ping
```

The property definition are as follows:

- **USER**: Defines the username of the user logging in.
- **PASSWORD**: The user password
- **HOST**: The host server to connect
- **XML_REQUEST**: The XML request that needs to be sent (ping in this case)
- **LOGIN_REQUIRED**: If true, a login is done before sending the XML_REQUEST.
- **URI**: Optional URI parameter that indicates the service that needs to be invoked. If this is not specified in the property file, it must be passed in as a java program argument.

Please note that the property values in the client.properties file like HOST, USER, PASSWORD, other values in the XML_REQUEST and the URI itself may need to modified to suit your deployment and the sample that is being executed. Edit this file appropriately before executing any of programs mentioned below.
8.1 CSM API pre-configuration checks

The following sample program implemented in java demonstrates a REST client program that checks if the CSM API is enabled for use. Use the client.properties file as defined above.

After compiling, use the following command to run the program:

```
Command Prompt> java RestClient <path_to_client.properties> [<uri>]
```

**Class RestClient.java**

```java
/**
 * Sample Program to test if CSM server is correctly configured for API
 */
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import java.io.FileInputStream;
import java.util.Properties;
public class RestClient {
    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;
```
static{
  initSSL();
}

private static void initSSL() {
    SSLContext sslContext = null;
    try {
        sslContext = SSLContext.getInstance("SSL");
        sslContext.init(null, new TrustManager[]{new X509TrustManager() {
            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
            public void checkClientTrusted(X509Certificate[] certs, String authType) {
                System.out.println("checkClientTrusted =============");
            }
            public void checkServerTrusted(X509Certificate[] certs, String authType) {
                System.out.println("checkServerTrusted =============");
            }
        }}, new SecureRandom());
        SSLSocketFactory sf = new SSLSocketFactory(sslContext);
        Scheme httpsScheme = new Scheme("https", 443, sf);
        SchemeRegistry schemeRegistry = new SchemeRegistry();
        schemeRegistry.register(httpsScheme);
        ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
        HttpClient httpclient = new DefaultHttpClient(cm, params);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (KeyManagementException e) {
        e.printStackTrace();
    }
}

/**
 * This method will send the XML payload and return the XML response as a string.
 * @param uri
 * @param host
 * @param isCookieNeeded
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascookie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}
```java
public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));

    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Element element = (Element) errorNodes.item(i);
        if (element.getTextContent() != null && !element.getTextContent().equals("")) {
            NodeList nodes = doc.getDocumentElement().getElementsByTagNames("description");
            for (int j = 0; j < nodes.getLength(); j++) {
                Element element2 = (Element) nodes.item(j);
                throw new Exception(element2.getTextContent());
            }
        }
    }

    if (response != null && response.trim().length() != 0) {
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println(response);
                //Do something...
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
                System.out.println(response);
                //Do something...
            } else {
                System.out.println("Some issue, Obtained HTTP status code ":"+sl.getStatusCode());
                System.out.println(response);
                //Do something...
            }
        }
    }
}
/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args){
    try {
        //Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);

        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML_REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        //If URI is not passed on command line see if its defined in properties file
        URI uri = new URI((args.length == 2)?args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        boolean autoLogin = false;
        if (null != temp && temp.trim().length() != 0) {
            autoLogin = Boolean.valueOf(temp);
        }

        RestClient client = new RestClient();
        if (uri.toString().endsWith("login")){
```
client.doPost(uri, payload, host, false);
} else {
    //Step 1 :
    if (autoLogin) {
        String login_payload = "<?xml version="1.0" encoding="UTF-8"?>
<loginRequest
xmlns:cm="cm">
<protVersion>1.0</protVersion>
<reqId>123</reqId>
<username>"+username+"</username>
<password>"+password+"</password>
</loginRequest>
    
    client.doPost(new URI("https://"+host+"/nbi/login"), login_payload, host, false);
    }
    //Step 2: 
    client.doPost(uri, payload, host, true);
    
    } catch (Exception ex) {
        System.out.println(ex.getMessage()); usage();
    }
}

public static void usage(){
    System.out.println("Please check the data entered in the properties file");
    System.out.println("Usage : ");
    System.out.println("java RestClient <path_to_client.properties> [<uri>]");
}

8.2 Login and ping test

The following simple sample program implemented in java demonstrates a REST client that logs in to the CSM server using the CSM API and then makes a “ping” request. Use the client.properties file as defined earlier.

After compiling, use the following command to run the program:

Command Prompt> java RestClient <path_to_client.properties> [<uri>]

Class RestClient.java

/**
   * Sample Program to login to the CSM Server and send a ping request
   */
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import org.apache.http.HttpStatus;
import org.apache.http.ParseException;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import java.io.FileInputStream;
import java.util.Properties;

public class RestClient {
    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;

    static{
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[] { new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers " + "---------");
                    return null;
                }
                public void checkClientTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkClientTrusted " + "---------");
                }
                public void checkServerTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkServerTrusted " + "---------");
                }
            }}, new SecureRandom());
            SSLSocketFactory sf = new SSLSocketFactory(sslContext);
            Scheme httpsScheme = new Scheme("https", 443, sf);
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(httpsScheme);
            HttpParams params = new BasicHttpParams();
            ClientConnectionManager cm = new ThreadSafeClientConnManager(params,
            schemeRegistry);
            httpclient = new DefaultHttpClient(cm, params);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (KeyManagementException e) {
            e.printStackTrace();
        }
    }

    /**
    * This method will send the XML payload and return the XML response as a string.
    * @param url
    * @param PayLoad
    */
}

Cisco Security Manager 4.4 API Specification (Version 1.1)
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws ClientProtocolException, IOException {

    HttpResponse httpresponse = null;
   HttpPost httppost = new HttpPost (uri);

    if (isCookieNeeded) {
        httpclient.setCookieStore(ascokie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws ParseException, IOException {

    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println(response);
                //Do something...
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
                System.out.println(response);
                //Do something...
            } else {
                System.out.println("Some issue, Obtained HTTP status code :"+sl.getStatusCode());
                System.out.println(response);
                //Do something...
            }
        }
    }
}

/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args){

try{
  //Load the basic properties
  FileInputStream fis = null;
  Properties prop = new Properties();
  fis = new FileInputStream(args[0]);
  prop.load(fis);

  String host = prop.getProperty("HOST");
  String payload = prop.getProperty("XML_REQUEST");
  String username = prop.getProperty("USER");
  String password = prop.getProperty("PASSWORD");


```java
String path = prop.getProperty("URI");
//If URI is not passed on commandline see if its defined in properties file
URI uri = new URI((args.length == 2) ? args[1] : path);
String temp = prop.getProperty("LOGIN_REQUIRED");
boolean autoLogin = false;
if(null != temp && temp.trim().length() != 0){
    autoLogin = Boolean.valueOf(temp);
}

RestClient client = new RestClient();
if(uri.toString().endsWith("login")){
    client.doPost(uri, payload, host, false);
}else{
    //Step 1:
    if(autoLogin){
        String login_payload = "<?xml version="1.0" encoding="UTF-8"?>
<loginRequest xmlns:csm="csm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <protVersion>1.0</protVersion>
    <reqId>123</reqId>
    <username>"+username+
    <password>"+password+"</password>
</loginRequest>"
    client.doPost(new URI("https://"+host+"/nbi/login"), login_payload, host, false);
    }
    //Step 2:
    client.doPost(uri, payload, host, true);
}
}

public static void usage(){
    System.out.println("Please check the data entered in the properties file");
    System.out.println("Usage : ");
    System.out.println("java RestClient <path_to_client.properties> [<uri>]\n");
}
```

### 8.3 Fetch CLI configuration of a firewall

The following simple sample program implemented in java demonstrates a REST client program that uses the CSM API to fetch the raw CLI configuration of a firewall from CSM’s database. Use the following client.properties as input (change the name of the device to match the device name in the server’s inventory) :

```
USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST=<xml version="1.0" encoding="utf-8"?>
<deviceConfigByNameRequest xmlns="csm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <protVersion>1.0</protVersion>
    <reqId>123</reqId>
    <name>firewall_device</name>
</deviceConfigByNameRequest>
```

# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/configservice/getDeviceConfigByName

After compiling, use the following command to run the program:

```
Command Prompt> java RestClient <path_to_client.properties> [uri]

Class RestClient.java

/**<n
 * Sample Program to get entire CLI of a firewall
 */

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import java.io.FileInputStream;
import java.util.Properties;

public class RestClient {

    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;

    static{
        initSSL();
    }

    private static void initSSL() {
        SSLContext ssslContext = null;
        try {
            ssslContext = SSLContext.getInstance("SSL");
            ssslContext.init(null, new TrustManager[]{ new X509TrustManager()
                { public X509Certificate[] getAcceptedIssuers() {
                        System.out.println("getAcceptedIssuers =============");
                    }
            });

            httpclient = new DefaultHttpClient(ssslContext);
        }
    }
}
return null;
}
public void checkClientTrusted(X509Certificate[] certs, String authType) {
    System.out.println("checkClientTrusted " + new SecureRandom());
}
public void checkServerTrusted(X509Certificate[] certs, String authType) {
    System.out.println("checkServerTrusted " + new SecureRandom());
}
}

SSLSocketFactory sf = new SSLSocketFactory(sslContext);
/
Scheme httpsScheme = new Scheme("https", 443, sf);
SchemeRegistry schemeRegistry = new SchemeRegistry();
HttpParams params = new BasicHttpParams();
ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
DefaultHttpClient httpclient = new DefaultHttpClient(cm, params);
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
} catch (KeyManagementException e) {
    e.printStackTrace();
}
/**
 * This method will send the XML payload and return the XML response as a string.
 * @param uri
 * @param host
 * @param isCookieNeeded
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascookie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    Document doc = domb.parse(new ByteArrayInputStream(response.getBytes()));
    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Element element = (Element) errorNodes.item(i);
        if (element.getTextContent() != null && !element.getTextContent().equals("")) {
            NodeList nodes = doc.getDocumentElement().getElementsByTagName("description");
            for (int j = 0; j < nodes.getLength(); j++) {
                System.out.println(nodes.item(j).getTextContent());
            }
        }
    }
}
```java
for (int j = 0; j < nodes.getLength(); j++) {
    Element element2 = (Element) nodes.item(j);
    throw new Exception(element2.getTextContent());
}
}
if(response != null && response.trim().length() != 0){
    StatusLine sl;
    if ((sl = httpresponse.getStatusLine()) != null) {
        if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
            System.out.println("Hit Authorization exception");
            System.out.println(response);
            //Do something...
        } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
            System.out.println("The request is a success...");
            System.out.println(response);
            //Do something...
        } else{
            System.out.println("Some issue, Obtained HTTP status code: "+sl.getStatusCode());
            System.out.println(response);
            //Do something...
        }
    }
}
/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args){
    try{
        //Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);
        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        //If URI is not passed on commandline see if its defined in properties file
        URI uri = new URI((args.length == 2)?args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        boolean autoLogin = false;
        if(null != temp && temp.trim().length() != 0){
            autoLogin = Boolean.valueOf(temp);
        }
        RestClient client = new RestClient();
        if(uri.toString().endsWith("login")){
            client.doPost(uri, payload, host, false);
        } else{
            //Step 1 :
            if(autoLogin){
                String login_payload = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<loginRequest xmlns:csd=\"csm\"><protVersion>1.0</protVersion><reqId>123</reqId><username="+username+"</username>
<password="+password+"</password></loginRequest>";
                client.doPost(new URI("https://"+host+"/nbi/login"), login_payload, host, false);
            }
            //Step 2:
```
8.4 Executing show access-list on a firewall device

The following simple sample program implemented in Java demonstrates a REST client using the CSM API to execute a show access-list command on a firewall device. Use the following client.properties as input (change the deviceIP to match the IP of a valid device in the server’s inventory):

```properties
USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST=<xml version="1.0" encoding="UTF-8"?><csm:execDeviceReadOnlyCLICmdsRequest xmlns:csm="csm"> <protVersion>1.0</protVersion> <reqId>123</reqId> <deviceReadOnlyCLICmd><deviceIP>192.168.1.1</deviceIP><cmd>show</cmd><argument>access-list</argument></deviceReadOnlyCLICmd></csm:execDeviceReadOnlyCLICmdsRequest>

# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/utilservice/execDeviceReadOnlyCLICmds
```

After compiling, use the following command to run the program:

```
Command Prompt> java RestClient <path_to_client.properties> [<uri>]
```

**Class RestClient.java**

```java
/**
 * Sample program to execute a show access-list command on a firewall
 */
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
```
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import java.io.FileInputStream;
import java.util.Properties;
public class RestClient {

    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;

    static {
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[] { new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers =============");
                    return null;
                }
            } }, new SecureRandom());
            SSLSocketFactory sf = new SSLSocketFactory(sslContext);
            Scheme httpsScheme = new Scheme("https", 443, sf);
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(httpsScheme);
            HttpParams params = new BasicHttpParams();
            ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
            httpclient = new DefaultHttpClient(cm, params);
        } catch (Exception e) {
            System.out.println("Exception =============");
        }
    }
}

Cisco Security Manager 4.4 API Specification (Version 1.1)
```java
/**
 * This method will send the XML payload and return the XML response as a string.
 * @param uri
 * @param host
 * @param isCookieNeeded
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore (ascookie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity (strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute (httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString (ent);
    DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Element element = (Element) errorNodes.item(i);
        if (element.getTextContent() != null && !element.getTextContent().equals("")) {
            NodeList nodes = doc.getDocumentElement().getElementsByTagName("description");
            for (int j = 0; j < nodes.getLength(); j++) {
                Element element2 = (Element) nodes.item(j);
                throw new Exception(element2.getTextContent());
            }
        }
    }
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println(response);
                //Do something...
            }else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
                System.out.println(response);
                //Do something...
            }else{
                //Do something...
            }
        }
    }
```
```java
public static void main(String[] args) {
    try {
        // Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);

        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML_REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        // If URI is not passed on commandline see if its defined in properties file
        URI uri = new URI((args.length == 2) ? args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        boolean autoLogin = false;
        if (null != temp && temp.trim().length() != 0) {
            autoLogin = Boolean.valueOf(temp);
        }

        RestClient client = new RestClient();
        if (uri.toString().endsWith("login")) {
            client.doPost(uri, payload, host, false);
        } else {
            // Step 1 :
            if (autoLogin) {
                String login_payload = "<?xml version="1.0" encoding="UTF-8"?>
<csm:loginRequest
    xmlns:csm="csm" xmlns:csm="csm" xmlns:csm="csm"><protVersion>1.0</protVersion><reqId>123</reqId><username">" + username + "</username>" + password + "</password></csm:loginRequest>"
                client.doPost(new URI("https://" + host + ":/nbi/login"), login_payload, host, false);
            }
            // Step 2:
            client.doPost(uri, payload, host, true);
        }
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
```

```
public static void usage() {
    System.out.println("Please check the data entered in the properties file");
    System.out.println("Usage : ");
    System.out.println("java RestClient <path_to_client.properties> [<uri>]" );
}
```
8.5 Fetch CSM defined firewall policy

The following simple sample program implemented in Java demonstrates a REST client that fetches the CSM firewall policy as it is defined in the CSM UI. Use the following client.properties file (Change the gid value to match the GID of a device in the server’s inventory):

```
USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST=<xml version="1.0" encoding="UTF-8"?>
<xml:policyConfigByDeviceGIDRequest xmlns:xml="csm">
  <proVersion>1.0</proVersion>
  <reqId>123</reqId>
  <gid>00000000-0000-0000-0000-004294967307</gid>
  <policyType>DeviceAccessRuleFirewallPolicy</policyType>
</xml:policyConfigByDeviceGIDRequest>
```

# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/configservice/getPolicyConfigById

After compiling, use the following command to run the program:

```
Command Prompt> java RestClient <path_to_client.properties> [uri]
```

Class RestClent.java

```java
/**
 * Sample Program to get access rules defined on a firewall as it appears in the
 * CSM UI.
 */
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
```
```java
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import java.io.FileInputStream;
import java.util.Properties;

public class RestClient {
    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpClient;

    static {
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[]{
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers ===========");
                    return null;
                }
            },
            new SecureRandom());
            SSLSocketFactory sf = new SSLSocketFactory(sslContext);
            Scheme httpsScheme = new Scheme("https", 443, sf);
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(httpsScheme);
            HttpParams params = new BasicHttpParams();
            ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
            httpClient = new DefaultHttpClient(cm, params);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (KeyManagementException e) {
            e.printStackTrace();
        }
    }

    public static void sendXML(String xmlPayload, String uri, String host, boolean isCookieNeeded) {
        // XML sending logic
    }

    /**
     * This method will send the XML payload and return the XML response as a string.
     * @param uri
     * @param host
     * @param isCookieNeeded
     * @return
     */
```

Cisco Security Manager 4.4 API Specification (Version 1.1)
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascookie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    Entity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Node element = (Node) errorNodes.item(i);
        if (element.getNodeType() == Node.ELEMENT_NODE) {
            String elementContent = element.getTextContent();
            if (elementContent != null && !elementContent.equals("") ) {
                NodeList nodes = doc.getDocumentElement().getElementsByTagName("description");
                for (int j = 0; j < nodes.getLength(); j++) {
                    Node node = (Node) nodes.item(j);
                    if (node.getNodeType() == Node.ELEMENT_NODE) {
                        String nodeContent = node.getTextContent();
                        throw new Exception(nodeContent);
                    }
                }
            }
        }
    } 
}

if (response != null && response.trim().length() != 0) {
    StatusLine sl;
    if ((sl = httpresponse.getStatusLine()) != null) {
        if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
            System.out.println("Hit Authorization exception");
            System.out.println(response);
            //Do something...
        } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
            System.out.println("The request is a success...");
            System.out.println(response);
            //Do something...
        } else {
            System.out.println("Some issue, Obtained HTTP status code :" + sl.getStatusCode());
            System.out.println(response);
            //Do something...
        }
    }
}

/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args){
try{
    //Load the basic properties
    FileInputStream fis = null;
    Properties prop = new Properties();
    fis = new FileInputStream(args[0]);
    prop.load(fis);

    String host = prop.getProperty("HOST");
    String payload = prop.getProperty("XML_REQUEST");
    String username = prop.getProperty("USER");
    String password = prop.getProperty("PASSWORD");
    String path = prop.getProperty("URI");
    //If URI is not passed on commandline see if its defined in properties file
    URI uri = new URI((args.length == 2) ? args[1] : path);
    String temp = prop.getProperty("LOGIN_REQUIRED");
    boolean autoLogin = false;
    if(null != temp && temp.trim().length() != 0){
        autoLogin = Boolean.valueOf(temp);
    }

    RestClient client = new RestClient();
    if(uri.toString().endsWith("login")){
        client.doPost(uri, payload, host, false);
    }else{
        //Step 1:
        if(autoLogin){
            String login_payload = "<?xml version="1.0" encoding="UTF-8"?><csm:loginRequest xmlns:csm="csm"><protVersion>1.0</protVersion><reqId>123</reqId><username>"+username+"</username><password>"+password+"</password></csm:loginRequest>";
            client.doPost(new URI("https://"+host+"/nbi/login"), login_payload, host, false);
        }
        //Step 2:
        client.doPost(uri, payload, host, true);
    }
}catch(Exception ex){
    System.out.println(ex.getMessage()); usage();
}

public static void usage(){
    System.out.println("Please check the data entered in the properties file");
    System.out.println("Usage : ");
    System.out.println("java RestClient <path_to_client.properties> [<uri>]");
}

8.6 List shared policies assigned to all devices

The following simple sample program implemented in java demonstrates a REST client that iterates over CSM’s device inventory and lists the directly assigned shared policies on all devices. Use the following client.properties file:

USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST=<xml version="1.0" encoding="UTF-8"?>
<csm:deviceListByCapabilityRequest xmlns:csm="csm"/>
<protVersion>1.0<protVersion>
<reqId>123</reqId>
<deviceCapability>*</deviceCapability>
</csm:deviceListByCapabilityRequest>
<xml:REQUEST1><?xml version="1.0" encoding="UTF-8"?>
<protVersion>1.0</protVersion>
<reqId>123</reqId>
<gid>DEVICE_ID</gid>
</csm:policyListByDeviceGIDRequest>

# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/configservice/getDeviceListByType
URI1=https://localhost/nbi/configservice/getPolicyListByDeviceGID

After compiling, use the following command to run the program:

Command Prompt> java RestClient <path_to_client.properties> [<uri>]

Class RestClient.java

/**
 * Sample program to collect the Shared Access Rules applied to Devices in CSM.
 */
import java.io.*;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import java.util.ArrayList;
import java.util.Properties;

public class RestClient {

    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;

    static{
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[] { new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers =========");
                    return null;
                }
                public void checkClientTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkClientTrusted =============");
                }
                public void checkServerTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkServerTrusted =============");
                }
            }}, new SecureRandom());
            SSLSocketFactory sf = new SSLSocketFactory(sslContext);
            Scheme httpsScheme = new Scheme("https", 443, sf);
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(httpsScheme);
            HttpParams params = new BasicHttpParams();
            ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
            httpclient = new DefaultHttpClient(cm, params);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (KeyManagementException e) {
            e.printStackTrace();
        }
    }

    /**
     * This method will send the XML payload and return the XML response as a string.
     * @param uri
     * @param host
     * @param isCookieNeeded
     * @return
     * @throws IOException
     * @throws ClientProtocolException
     */
    public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
        HttpResponse httpresponse = null;
        HttpPost httppost = new HttpPost (uri);
        if (isCookieNeeded) {
            httpclient.setCookieStore(ascookie);
        }
        httppost.addHeader("Content-Type", "text/xml");
        StringEntity strEntity = new StringEntity (payload, "UTF-8");
        httppost.setEntity(strEntity);
    }
}
```java
public void processResponse (HttpServletResponse httpResponse) throws Exception, IOException, SAXException {
    HttpEntity entity = httpResponse.getEntity();
    String response = EntityUtils.toString(entity);
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpResponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                //Do something...
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
            }
        } else {
            System.out.println("Some issue, Obtained HTTP status code: " + sl.getStatusCode());
            System.out.println(response);
            //Do something...
        }
    }
}

/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args){
    try {
        //Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);
        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML_REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        //If URI is not passed on commandline see if its defined in properties file
        URI uri = new URI((args.length == 2)?args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        String policy_req = prop.getProperty("XML_REQUEST1");
        String policy_req_path = prop.getProperty("URI1");
        boolean autoLogin = false;
        if (null != temp && temp.trim().length() != 0){
            autoLogin = Boolean.valueOf(temp);
        }

        RestClient client = new RestClient();
        if (uri.toString().endsWith("login")){
            client.doPost(uri, payload, host, false);
        } else {
            //Step 1 :
            if (autoLogin)
```
String login_payload = "<?xml version="1.0" encoding="UTF-8"?>
<loginRequest
xmlns:csm="csm">
<protVersion>1.0</protVersion>
<reqId>123</reqId>
<username>" + username + "</username>
<password>" + password + "</password>
</loginRequest>";
client.doPost(new URI("https://" + host + "/nbi/login"), login_payload, host, false);

// Step 2: Get All Devices
ArrayList devices = client.getDeviceList(uri, payload, host, true);

// Step 3: Get Shared Policies on Devices.
ArrayList sharedAccessRules = new ArrayList();
for (int i = 0; i < devices.size(); i++) {
    String device = (String) devices.get(i);
    String policy_request = policy_request.replace("DEVICE_ID", device);
    ArrayList policies = client.getPolicyList(new URI(policy_req_path),
policy_request, host, true);
    if (policies!= null) {
        sharedAccessRules.addAll(policies);
    }
    System.out.println(sharedAccessRules);
}
} catch (Exception ex){
    System.out.println(ex.getMessage());
}

private ArrayList getPolicyList(URI uri, String payload, String host, boolean isCookieNeeded)
throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascokie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : " + uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    //processResponse(httpresponse);
    return getPolicyList(httpresponse);
}

private ArrayList getPolicyList(HttpResponse httpresponse) throws IOException, SAXException,
ParserConfigurationException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    System.out.println(response);
    ArrayList<String> retArr = new ArrayList();
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println(response);
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
                DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
                Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
        } catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("policyDesc");
for (int i = 0; i < errorNodes.getLength(); i++) {
    Element element = (Element) errorNodes.item(i);
    String text = element.getTextContent();
    if (text != null && !text.equals(""))
        text.indexOf("DeviceAccessRuleFirewallPolicy") >= 0) {
            text = text.replaceAll("DeviceAccessRuleFirewallPolicy", "");
            if (!text.equals("-- local --")) // Filter out Local rules.
                retArr.add(text);
        }
} else {
    System.out.println("Some issue, Obtained HTTP status code:");
    System.out.println(response);
    //Do something...
}
}
return retArr;

private ArrayList getDeviceList(URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
   HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascookie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    //processResponse(httpresponse);
    return getDeviceList(httpresponse);
}

private ArrayList getDeviceList(HttpResponse httpresponse) throws IOException, SAXException, ParserConfigurationException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    ArrayList<String> retArr = new ArrayList();
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println(response);
                //Do something...
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
                DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
                Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
                NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("gid");
                for (int i = 0; i < errorNodes.getLength(); i++) {
                    Element element = (Element) errorNodes.item(i);
                    if (element.getTextContent() != null && !element.getTextContent().equals(""))
                    {
                        retArr.add(text.replaceAll("DeviceAccessRuleFirewallPolicy", "");
                        if (!text.equals("-- local --")) // Filter out Local rules.
                            retArr.add(text);
                    }
                }
            }
            }
        System.out.println("Some issue, Obtained HTTP status code:");
        System.out.println(response);
        //Do something...
    }
    return retArr;
}
8.7 List content of a given shared policy

The following simple sample program implemented in java demonstrates a REST client the lists the content of a firewall access rule shared policy given the name of the shared policy. Use the client.properties file as defined below (change shared policy name defined in the properties file appropriately):

```xml
<xml version="1.0" encoding="UTF-8">
<protVersion>1.0</protVersion>
</xml>
```

After compiling, use the following command to run the program:

```
Command Prompt> java RestClient <path_to_client.properties> [<uri>]
```

**Class RestClient.java**

```java
/**
 * Sample program to get the contents of a shared policy given the shared policy name.
 */
import java.io.ByteArrayInputStream;
```
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;

import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import java.io.FileInputStream;
import java.util.Properties;

public class RestClient {
    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpClient;

    static {
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[] { new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers =============");
                    return null;
                }
                public void checkClientTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkClientTrusted =============");
                }
                public void checkServerTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkServerTrusted =============");
                }
            }}, new SecureRandom());
        } catch (NoSuchAlgorithmException | KeyManagementException e) {
            System.out.println(e.getMessage());
        }
    }
}
SSLsocketFactory sf = new SSLsocketFactory(sslContext);

Scheme httpsScheme = new Scheme("https", 443, sf);
SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(httpsScheme);
HttpParams params = new BasicHttpParams();
ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
httpclient = new DefaultHttpClient(cm, params);
}
catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
catch (KeyManagementException e) {
    e.printStackTrace();
}
}
/**
 * This method will send the XML payload and return the XML response as a string.
 * @param uri
 * @param host
 * @param isCookieNeeded
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {
    HttpResponse httpresponse = null;
    HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpClient.setCookieStore(ascokie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    HttpEntity ent = httpresponse.getEntity();
    DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Element element = (Element) errorNodes.item(i);
        if(element.getTextContent() != null && !element.getTextContent().equals("")) {
            NodeList nodes = doc.getDocumentElement().getElementsByTagName("description");
            for (int j = 0; j < nodes.getLength(); j++) {
                Element element2 = (Element) nodes.item(j);
                throw new Exception(element2.getTextContent());
            }
        }
    }
    if(response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {

Cisco Security Manager 4.4 API Specification (Version 1.1)
```java
System.out.println("Hit Authorization exception");
System.out.println(response);
// Do something...
else if (sl.getStatusCode() == HttpStatus.SC_OK) {
    System.out.println("The request is a success...");
    System.out.println(response);
    // Do something...
} else {
    System.out.println("Some issue, Obtained HTTP status code:");
    System.out.println(response);
    // Do something...
}
}

/**
 * Main method processing the request/response
 * @param args
 */
public static void main(String[] args) {
    try {
        // Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);
        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML_REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        // If URI is not passed on commandline see if its defined in properties file
        URI uri = new URI((args.length == 2)?args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        boolean autoLogin = false;
        if (null != temp && temp.trim().length() != 0) {
            autoLogin = Boolean.valueOf(temp);
        }
        RestClient client = new RestClient();
        if (uri.toString().endsWith("login")) {
            client.doPost(uri, payload, host, false);
        } else {
            // Step 1: if (autoLogin) {
            String login_payload = "<csm:loginRequest xmlns:csm="csm">
            <protVersion>1.0</protVersion>
            <reqId>123</reqId>
            <username>" + username + "</username>
            <password>" + password + "</password>
            </csm:loginRequest>";
            client.doPost(new URI("https://" + host + "/nbi/login"), login_payload, host, false);
        }
        // Step 2:
        client.doPost(uri, payload, host, true);
    }
} catch (Exception ex) {
    System.out.println(ex.getMessage());
    usage();
}

public static void usage() {
    System.out.println("Please check the data entered in the properties file");
    System.out.println("Usage :");
    System.out.println("java RestClient <path_to_client.properties> [<uri>]")
```
8.8 Subscribing to change notifications – Deployment, OOB

The following simple sample program implemented in java demonstrates a REST client that registers for change notifications from CSM to receive Deployment and Out of Band (OOB) events. Please note that to receive change notifications, the client registers a specific syslog service (IP and Port) to which CSM will send ‘asynchronous’ notifications. To see the actual notifications, please use any open source Syslog server or implement a simple UDP listener at the specified IP/Port (see SyslogServer element below) to list the notification content.

Use the client.properties file as listed below:

```properties
USER=admin
PASSWORD=admin
HOST=localhost
XML_REQUEST= <?xml version="1.0" encoding="UTF-8"?>
<op>add</op>
<subscriptionId>123454</subscriptionId>
<eventFilterItem>
<filterEventType>syslog</filterEventType>
<filterEventFormat>xml</filterEventFormat>
<filterEventCategory>configChange</filterEventCategory>
</eventFilterItem>
<syslogServer>
<port>514</port>
<destAddress>10.10.10.10</destAddress>
</syslogServer>
</csm:eventSubRequest>
```

# Set LOGIN_REQUIRED to true if the URI supplied
# requires login to be done as a prerequisite.
LOGIN_REQUIRED=true
URI=https://localhost/nbi/eventservicen
eventSubscription

After compiling, use the following command to run the program:

Command Prompt> java RestClient <path_to_client.properties> []<uri>

Class RestClient.java

```java
/**
 * Sample program to subscribe to change events and get more details on latest change.
 */
import java.io.ByteArrayInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URI;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
```
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import java.util.Properties;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.http.HttpStatus;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.util.EntityUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class RestClient {

    public static CookieStore ascookie = null;
    public static DefaultHttpClient httpclient;

    static {
        initSSL();
    }

    private static void initSSL() {
        SSLContext sslContext = null;
        try {
            sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, new TrustManager[]{new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers =============");
                    return null;
                }
                public void checkClientTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkClientTrusted =============");
                }
                public void checkServerTrusted(X509Certificate[] certs, String authType) {
                    System.out.println("checkServerTrusted =============");
                }
            }}, new SecureRandom());

            SSLSocketFactory sf = new SSLSocketFactory(sslContext);
            Scheme httpsScheme = new Scheme("https", 443, sf);
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(httpsScheme);
        }
    }
}
HttpParams params = new BasicHttpParams();
ClientConnectionManager cm = new ThreadSafeClientConnManager(params, schemeRegistry);
httpclient = new DefaultHttpClient(cm, params);
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
} catch (KeyManagementException e) {
    e.printStackTrace();
}
/**
 * This method will send the XML payload and return the XML response as a string.
 * @param uri
 * @param host
 * @param isCookieNeeded
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public void doPost (URI uri, String payload, String host, boolean isCookieNeeded) throws Exception {

    HttpResponse httpresponse = null;
   HttpPost httppost = new HttpPost (uri);
    if (isCookieNeeded) {
        httpclient.setCookieStore(ascokie);
    }
    httppost.addHeader("Content-Type", "text/xml");
    StringEntity strEntity = new StringEntity (payload, "UTF-8");
    httppost.setEntity(strEntity);
    System.out.println("Calling : "+uri.toString());
    httpresponse = httpclient.execute(httppost);
    ascookie = httpclient.getCookieStore();
    processResponse(httpresponse);
}

public void processResponse (HttpResponse httpresponse) throws Exception, IOException, SAXException {
    HttpEntity ent = httpresponse.getEntity();
    String response = EntityUtils.toString(ent);
    DocumentBuilder domp = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = domp.parse(new ByteArrayInputStream(response.getBytes()));
    NodeList errorNodes = doc.getDocumentElement().getElementsByTagName("code");
    for (int i = 0; i < errorNodes.getLength(); i++) {
        Element element = (Element) errorNodes.item(i);
        if (element.getTextContent() != null && !element.getTextContent().equals("")) {
            NodeList nodes = doc.getDocumentElement().getElementsByTagName("description");
            for (int j = 0; j < nodes.getLength(); j++) {
                Element element2 = (Element) nodes.item(j);
                throw new Exception(element2.getTextContent());
            }
        }
    }
    if (response != null && response.trim().length() != 0){
        StatusLine sl;
        if ((sl = httpresponse.getStatusLine()) != null) {
            if (sl.getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
                System.out.println("Hit Authorization exception");
                System.out.println("The request is a success...");
            } else if (sl.getStatusCode() == HttpStatus.SC_OK) {
                System.out.println("The request is a success...");
            }
        }
    }
}
```java
//Do something...
} else {
    System.out.println("Some issue, Obtained HTTP status code :" + sl.getStatusCode());
    System.out.println(response);
    //Do something...
}
}
/** *
 * Main method processing the request/response *
 * @param args *
 */
public static void main(String[] args) {
    try {
        //Load the basic properties
        FileInputStream fis = null;
        Properties prop = new Properties();
        fis = new FileInputStream(args[0]);
        prop.load(fis);

        String host = prop.getProperty("HOST");
        String payload = prop.getProperty("XML_REQUEST");
        String username = prop.getProperty("USER");
        String password = prop.getProperty("PASSWORD");
        String path = prop.getProperty("URI");
        //If URI is not passed on commandline see if its defined in properties file
        URI uri = new URI((args.length == 2)?args[1] : path);
        String temp = prop.getProperty("LOGIN_REQUIRED");
        boolean autoLogin = false;
        if (null != temp && temp.trim().length() != 0) {
            autoLogin = Boolean.valueOf(temp);
        }
        RestClient client = new RestClient();
        if (uri.toString().endsWith("login")) {
            client.doPost(uri, payload, host, false);
        } else {
            //Step 1 :
            if (autoLogin) {
                String login_payload = "<csm:loginRequest xmlns:csm="csm" xmlns:cm="cm"><protVersion>1.0</protVersion><reqId>123</reqId><username>" + username + "</username><password>" + password + "</password></csm:loginRequest>";
                client.doPost(new URI("https://" + host + "/nbi/login"), login_payload, host, false);
            }
            //Step 2:
            client.doPost(uri, payload, host, true);
        }
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
        usage();
    }

    public static void usage() {
        System.out.println("Please check the data entered in the properties file");
        System.out.println("Usage : ");
        System.out.println("java RestClient <path_to_client.properties> [<uri>]");
    }
}
```

Cisco Security Manager 4.4 API Specification (Version 1.1)
9 Troubleshooting (Common Scenarios)

Symptom: Certificate errors when communicating with the server
The CSM server uses self-signed certificates. Change the client program to accept self-signed certificates. Also API access is only available over HTTPS and HTTP access is disabled for security reasons.

Symptom: Authentication errors even after a successful login
Intermittent session errors will be seen if the session has timed out or has been invalidated. Default session inactivity timeout is 15 minutes. The session can be kept alive using a ping or heartbeat mechanism.

Symptom: Inline IP entered in the CSM GUI are retrieved as objects
CSM automatically encapsulates certain inline values (IP’s, Interface names etc.) inside a policy object for convenience.

Symptom: Data visible in the CSM client UI are not seen in the response data when fetched via the API
This can be due to various reasons. Some of them are listed below:

- API only returns committed data (i.e. all changes must be submitted). So check if all data that is visible in the client is committed.
- In some cases CSM specific settings are not returned in the response as it may not be relevant for API access.
- Check if the user has sufficient Role Based Access Control (RBAC) privileges to access the data

Symptom: Client does not receive any event notifications from the server
Some of the reasons for this problem:

- The event subscription has not been done or has been registered with incorrect syslog server that receives the notification
- Session used to register the event subscription has timed out. All event notifications are tied to a user session. If the user session is invalidated then all event notifications for that user session will be stopped.
10 XML Schema

The XML schema is broken into four files.

10.1 Common XSD

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:csm="csm" targetNamespace="csm">
    <xs:simpleType name="ObjectIdentifier">
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="ObjectIdentifierList">
        <xs:sequence/>
    </xs:complexType>

    <xs:complexType name="BaseObject">
        <xs:sequence>
            <xs:element name="gid" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
            <xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="lastUpdateTime" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
            <xs:element name="parentGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
            <xs:element name="updatedByUser" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="lastCommitTime" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
            <xs:element name="ticketId" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="activityName" type="xs:string" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="BaseError">
        <xs:sequence>
            <xs:element name="code" type="xs:unsignedLong" minOccurs="1" maxOccurs="1"/>
            <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="BaseReqResp">
        <xs:sequence>
            <xs:element name="protVersion" type="xs:double" minOccurs="0" maxOccurs="1"/>
            <xs:element name="reqId" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="endIndx" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
            <xs:element name="totalCount" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
            <xs:element name="error" type="BaseError" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="EntityDescriptor">
        <xs:sequence>
            <xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Device">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence>
                    <xs:element name="osType" type="OSType" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="osVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="imageName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="sysObjectID" type="xs:string" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="fullConfig" type="xs:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```
<xs:element name="mgmtInterface" type="Interface" minOccurs="0" maxOccurs="1"/>
<xs:element name="interfaceList" type="InterfaceList" minOccurs="0" maxOccurs="1"/>
<xs:element name="virtualContextList" type="Device" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="configState" type="ConfigurationState" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:extension>
</xs:complexType>
<xs:complexType name="PortIdentifier">
<xs:sequence>
  <!-- for non-modular chassis or chassis with a continuous port numbering scheme slot/module are not included -->
  <xs:element name="slotNum" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
  <xs:element name="moduleNum" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
  <xs:element name="portNum" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ProtocolPort">
<xs:restriction base="xs:string"/>
</xs:complexType>
<xs:complexType name="InterfaceList">
<xs:sequence>
  <xs:element name="interface" type="Interface" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Interface">
<xs:sequence>
  <xs:element name="type" type="xs:string" minOccurs="1" maxOccurs="1"/>
  <xs:element name="identifier" type="xs:string" minOccurs="1" maxOccurs="1"/>
  <xs:element name="ipInterface" type="IPInterfaceAttrss" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="macInterface" type="MACInterfaceAttrss" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="MACInterfaceAttrss">
<xs:sequence>
  <xs:element name="macAddress" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="IPInterfaceAttrss">
<xs:sequence>
  <xs:element name="domainName" type="xs:string" minOccurs="0" maxOccurs="1"/>
  <xs:element name="ipAddress" type="xs:string" minOccurs="1" maxOccurs="1"/>
  <xs:element name="isNatAddress" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
  <xs:element name="realIpAddress" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="OSType">
<xs:restriction base="xs:token">
  <xs:enumeration value="ios"/>
  <xs:enumeration value="fwsrm"/>
  <xs:enumeration value="asa"/>
  <xs:enumeration value="ips"/>
  <xs:enumeration value="pix"/>
  <xs:enumeration value="undefined"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="IPTransportProtocol">
<xs:restriction base="xs:token">
  <xs:enumeration value="TCP"/>
  <xs:enumeration value="UDP"/>
  <xs:enumeration value="IP"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="ConfigurationState">
<xs:restriction base="xs:token">
  <xs:enumeration value="undefined"/>
</xs:restriction>
</xs:simpleType>
<xs:enumeration value="committed"/>
<xs:enumeration value="deployed"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="DeviceCapability">
<xs:restriction base="xs:token">
<xs:enumeration value="firewall"/>
<xs:enumeration value="ids"/>
<xs:enumeration value="router"/>
<xs:enumeration value="switch"/>
<xs:enumeration value="*"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="DeviceGroup">
<xs:complexContent/>
<xs:restriction base="xs:string">
<xs:sequence/>
</xs:restriction>
</xs:complexType>
<xs:simpleType name="DeviceGroupPath">
<xs:sequence/>
</xs:complexType>
<xs:simpleType name="SubscriptionOperation">
<xs:restriction base="xs:string">
<xs:enumeration value="add"/>
<xs:enumeration value="delete"/>
</xs:restriction>
</xs:simpleType>

<!-- Common Service Methods

-->
<xs:complexType name="HeartbeatCallbackRequest">
  <xs:complexContent base="BaseReqResp"/>
</xs:complexType>

<xs:complexType name="LogoutRequest">
  <xs:complexContent base="BaseReqResp"/>
</xs:complexType>

<xs:complexType name="LogoutResponse">
  <xs:complexContent base="BaseReqResp"/>
</xs:complexType>

<xs:complexType name="PingRequest">
  <xs:complexContent base="BaseReqResp"/>
</xs:complexType>

<xs:complexType name="PingResponse">
  <xs:complexContent base="BaseReqResp"/>
</xs:complexType>

<xs:complexType name="GetServiceInfoRequest">
  <xs:complexContent base="BaseReqResp">
    <xs:sequence>
      <xs:element name="serviceVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="serviceName" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="serviceDesc" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="GetServiceInfoResponse">
  <xs:complexContent base="BaseReqResp">
    <xs:extension base="BaseReqResp"/>
  </xs:complexContent>
</xs:complexType>
10.2 Config XSD

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:csm="targetNamespace="csm">
    <xs:include schemaLocation="common.xsd"/>

    <xs:complexType name="BasePolicy">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="configState" type="ConfigurationState" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="isProperty" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="subType" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="refGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="isGroup" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secTag" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="BasePolicyObject">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="comment" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="nodeGID" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="isMandatoryAggregation" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="isProperty" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="subType" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="refGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="isGroup" type="xsd:boolean" minOccurs="0" maxOccurs="1"/> string "maxOccurs="0" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secTag" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="NetworkObjectsRefs">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="networkObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="interfaceRoleObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="NetworkInterfaceObjectsRefs">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="ipv4Data" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="ipData" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="SecurityGrpObjectsRef">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="securityGrpObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secName" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secTag" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="SecurityGrpObjectsRefs">
        <xs:complexContent>
            <xs:extension base="BaseObject">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="securityGrpObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secName" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xs:element name="secTag" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:element name="securityTag" type="SecurityGrpObjectsRef" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="IdentityUserGrpObjectsRefs">
<xs:sequence>
  <xs:element name="IdentityUserGrpObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1"/>
  <xs:element name="userNameData" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="userGroupData" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NetworkObjectRefs">
<xs:sequence>
  <xs:element name="networkObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
  <xs:element name="ipv4Data" type="xs:string" minOccurs="0" maxOccurs="1"/>
  <xs:element name="ipData" type="xs:string" minOccurs="0" maxOccurs="1"/>
  <xs:element name="interfaceKeyword" type="xs:string" fixed="interface" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NetworkOrIPRef">
<xs:choice>
  <xs:element name="hostOrNetworkObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
  <xs:element name="ipv4Data" type="xs:string" minOccurs="1" maxOccurs="1"/>
  <xs:element name="ipData" type="xs:string" minOccurs="1" maxOccurs="1"/>
</xs:choice>
</xs:complexType>

<xs:complexType name="NetworkPolicyObject">
<xs:complexContent base="BasePolicyObject">
<xs:extension name="policyTag" type="SecurityGrpObjectsRef">
<xs:sequence>
  <xs:element name="networkObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
  <xs:element name="ipv4Data" type="xs:string" minOccurs="0" maxOccurs="1"/>
  <xs:element name="ipData" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="IdentityUserGroupPolicyObject">
<xs:complexContent base="BasePolicyObject">
<xs:extension name="policyTag" type="SecurityGrpObjectsRef">
<xs:sequence>
  <xs:element name="userNameData" type="xs:string" minOccurs="0" maxOccurs="1"/>
  <xs:element name="userGroupData" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="SecurityGroupPolicyObject">
<xs:complexContent base="BasePolicyObject">
<xs:extension name="policyTag" type="SecurityGrpObjectsRef">
<xs:sequence>
  <xs:element name="securityTag" type="SecurityGrpObjectsRef" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

---

Cisco Security Manager 4.4 API Specification (Version 1.1)
```xml
<xs:complexType>
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="startPort" type="PortIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="endPort" type="PortIdentifier" minOccura="1" maxOccur="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ServiceParameters">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="sourcePort" minOccurs="0" maxOccurs="1"/>
    <xs:complexType>
      <xs:choice>
        <xs:element name="port" type="ProtocolPort"/>
        <xs:element name="portRefID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
      </xs:choice>
    </xs:complexType>
    <xs:element name="destinationPort" minOccurs="0" maxOccurs="1"/>
    <xs:complexType>
      <xs:choice>
        <xs:element name="port" type="ProtocolPort"/>
        <xs:element name="portRefID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
      </xs:choice>
    </xs:complexType>
    <xs:element name="icmpMessage" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:sequence>
      <xs:complexType>
        <xs:extension base="BasePolicyObject">
          <xs:sequence minOccurs="0" maxOccurs="1">
            <xs:element name="serviceParameters" type="ServiceParameters" minOccurs="1" maxOccurs="1"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexType>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="InterfaceRolePolicyObject">
  <xs:complexType>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="pattern" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:complexType>

<xs:complexType name="TimeRangePolicyObject">
  <xs:complexType>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="startDateTime" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
      <xs:element name="timeRange" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:complexType>
      <xs:choice>
        <xs:element name="dayOfWeekInterval"/>
      </xs:complexType>
    </xs:complexType>
  </xs:complexType>
</xs:complexType>
```

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:complexType name="SLAMonitorPolicyObject">
  <xs:complexContent>
    <xs:extension base="BasePolicyObject">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="slaId" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>
        <xs:element name="interfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
        <xs:element name="monitoredAddress" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="dataSizeInBytes" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="thresholdInMillis" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="timeoutInMillis" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="frequencyInSeconds" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="toS" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="numberOIPackets" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:element name="serviceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
<xs:element name="doLogging" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="logInterval" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="logLevel" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="logOption" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="permit" type="xs:boolean" minOccurs="1" maxOccurs="1"/>

<xs:extension>
  <xs:complexContent>
    <xs:restriction base="BasePolicyObject">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="references" minOccurs="1" maxOccurs="1" />
        <xs:element name="sequenceNumber" type="xs:unsignedInt" minOccurs="1" maxOccurs="1" />
        <xs:choice>
          <xs:element name="aclObjectReferenceGID" type="ObjectIdentifier" />
          <xs:element name="aceReferenceGID" type="ObjectIdentifier" />
        </xs:choice>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:extension>

<xs:complexType name="DeviceAccessRuleFirewallPolicy">
  <xs:complexContent>
    <xs:restriction base="BasePolicy">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="isEnabled" type="xs:boolean" minOccurs="0" maxOccurs="1" />
        <xs:element name="direction" minOccurs="0" maxOccurs="1" />
        <xs:element name="serviceObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1" />
        <xs:element name="users" type="IdentityUserGrpObjectsRefs" minOccurs="0" maxOccurs="1" />
        <xs:element name="sources" type="NetworkInterfaceObjectsRefs" minOccurs="0" maxOccurs="1" />
        <xs:element name="destinations" type="NetworkInterfaceObjectsRefs" minOccurs="0" maxOccurs="1" />
        <xs:element name="services" minOccurs="0" maxOccurs="1" />
        <xs:element name="serviceObjectGIDs" type="ObjectIdentifierList" minOccurs="0" maxOccurs="1" />
        <xs:element name="serviceParameters" type="ServiceParameters" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="logOptions" minOccurs="0" maxOccurs="1" />
        <xs:element name="isFirewallLoggingEnabled" type="xs:boolean" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:choice>
  <xs:element name="isDefaultLogging" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
  <xs:sequence minOccurs="0" maxOccurs="1">
    <xs:element name="loggingInterval" type="xs:unsignedInt"/>
    <xs:element name="loggingLevel" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="isIOSLoggingEnabled" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
    <xs:element name="isLogInput" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:choice>

<xs:complexType name="DeviceAccessRuleUnifiedFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="DeviceAccessRuleFirewallPolicy">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="sourceSG" type="SecurityGrpObjectsRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="destinationSG" type="SecurityGrpObjectsRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="deviceStaticRouting" type="DeviceStaticRoutingFirewallPolicy" minOccurs="0" maxOccurs="1"/>
        <xs:element name="deviceStaticRoutingRouterPolicy" type="DeviceStaticRoutingRouterPolicy" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceStaticRoutingFirewallPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="interfaceGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
        <xs:element name="networks" type="NetworkObjectsRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="gateway" type="NetworkObjectRefs" minOccurs="0" maxOccurs="1"/>
        <xs:element name="metric" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="tunnelled" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
        <xs:element name="tunnelled" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceStaticRoutingRouterPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="destinationNetwork" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:complexType name="DeviceBGPRouterPolicy">
  <xs:complexContent>
    <xs:extension base="BasePolicy">
      <xs:sequence>
        <xs:element name="networks" type="NetworkObjectsRefs" maxOccurs="unbounded" minOccurs="1"/>
        <xs:element name="networks" type="NetworkObjectsRefs" maxOccurs="unbounded" minOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
maxOccurs="1"/>

  </xs:element>
  </xs:choice>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="portRedirection" minOccurs="0" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="protocol" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="localPort" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>

    </xs:sequence>
  </xs:complexType>
</xs:element>

maxOccurs="1"/>

<xs:element name="globalPort" type="xs:unsignedInt" minOccurs="1" maxOccurs="1"/>

<xs:sequence>
  </xs:complexType>
  </xs:element>
</xs:complexType>

<xs:complexType name="InterfaceNATDynamicRulesRouterPolicy">
  <xs:extension base="BasePolicy">
    <xs:sequence>
      <xs:element name="trafficFlowAclObjectGID" type="objectIdentifier" minOccurs="1" maxOccurs="1"/>

      <xs:element name="translated" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:choice>
            <xs:element name="interfaceGID" type="objectIdentifier" minOccurs="1" maxOccurs="1"/>

            <xs:element name="addressPool" type="xs:string" minOccurs="1" maxOccurs="1"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>

      <xs:element name="unbounded"/>
    </xs:sequence>

    <xs:extension>
      <xs:complexType>
        <xs:element name="settings" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="enablePortTrans" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
              <xs:element name="noTransVPN" type="xs:boolean" minOccurs="1" maxOccurs="1"/>

              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:complexType>
      </xs:extension>
    </xs:extension>
  </xs:complexType>
</xs:element>

<xs:complexType name="DeviceNATTimeoutsRouterPolicy">
  <xs:complexContent>
    <xs:restriction base="xs:string">
      <xs:simpleContent>
        <xs:extension base="DeviceNATTimeoutsRouterPolicy">
          <xs:complexContent>
            <xs:restriction base="DeviceNATTimeoutsRouterPolicy">
              <xs:sequence>
                <xs:element name="noTransVPN" type="xs:boolean" minOccurs="1" maxOccurs="1"/>

                <xs:element name="unbounded"/>
              </xs:sequence>

              <xs:extension>
                <xs:complexType>
                  <xs:element name="settings" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="enablePortTrans" type="xs:boolean" minOccurs="1" maxOccurs="1"/>

                        <xs:element name="trafficFlowAclObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>

                        <xs:element name="translated" minOccurs="1" maxOccurs="1">
                          <xs:complexType>
                            <xs:choice>
                              <xs:element name="interfaceGID" type="objectIdentifier" minOccurs="1" maxOccurs="1"/>

                              <xs:element name="addressPool" type="xs:string" minOccurs="1" maxOccurs="1"/>
                            </xs:choice>
                          </xs:complexType>
                        </xs:element>

                        <xs:element name="unbounded"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:complexType>
              </xs:extension>
            </xs:complexContent>
          </xs:restriction>
        </xs:extension>
      </xs:complexContent>
    </xs:restriction>
  </xs:complexType>
</xs:element>
<xs:extension base="BasePolicy">
  <xs:sequence>
    <xs:element name="maxEntriesInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="timeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="udpTimeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="dnsTimeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="tcpTimeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="finRstTimeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="icmpTimeoutInSecs" type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
    <xs:element name="ipTcpConnPerRule" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
    <xs:element name="maxTCPConnPerRule" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:complexType>
    <xs:complexContent>
      <xs:sequence>
        <xs:element name="maxTCPConnPerRule" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element name="maxUDPConnPerRule" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexContent>
  </xs:complexType>
</xs:extension>

Firewall NAT

Reusable Firewall advanced options

Firewall NAT Advanced Options

Reusable NAT Type

NAT Type

Reusable NAT Type

Reusable Protocol Type

Interface NAT Address Pool Firewall Policy

Device NAT Trans Options Firewall Policy

Interface NAT Trans Exemptions Firewall Policy

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:element name="isExempt" type="xs:boolean" minOccurs="1" maxOccurs="1" />
<xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1" />
<xs:element name="original" type="NetworkInterfaceObjectsRefs" minOccurs="1" maxOccurs="1" />
<xs:element name="outsideNAT" type="xs:boolean" minOccurs="1" maxOccurs="1" />
<xs:element name="destinations" type="NetworkInterfaceObjectsRefs" minOccurs="1" maxOccurs="1" maxOccurs="1" />
<xs:element name="wsmAdvancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" />

<xs:complexType name="InterfaceNATDynamicRulesFirewallPolicy">
  <xs:complexContent maxOccurs="1" minOccurs="1" />
  <xs:extension base="BasePolicy">
    <xs:sequence maxOccurs="1">
      <xs:element name="isRuleEnabled" type="xs:boolean" minOccurs="1" maxOccurs="1" />
      <xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1" />
      <xs:element name="poolId" type="xs:unsignedInt" minOccurs="1" maxOccurs="1" />
      <xs:element name="original" type="NetworkObjectsRefs" minOccurs="1" maxOccurs="1" />
      <xs:element name="outsideNAT" type="xs:boolean" minOccurs="1" maxOccurs="1" />
      <xs:element name="advancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" />
    </xs:sequence>
  </xs:extension>
</xs:complexType>

<xs:complexType name="InterfaceNATPolicyDynamicRulesFirewallPolicy">
  <xs:complexContent maxOccurs="1" minOccurs="1" />
  <xs:extension base="BasePolicy">
    <xs:sequence maxOccurs="1">
      <xs:element name="isRuleEnabled" type="xs:boolean" minOccurs="1" maxOccurs="1" />
      <xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1" />
      <xs:element name="poolId" type="xs:unsignedInt" minOccurs="1" maxOccurs="1" />
      <xs:element name="original" type="NetworkInterfaceObjectsRefs" minOccurs="1" maxOccurs="1" />
      <xs:element name="outsideNAT" type="xs:boolean" minOccurs="1" maxOccurs="1" />
      <xs:element name="advancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" />
    </xs:sequence>
  </xs:extension>
</xs:complexType>

<xs:complexType name="InterfaceNATStaticRulesFirewallPolicy">
  <xs:complexContent maxOccurs="1" minOccurs="1" />
  <xs:extension base="BasePolicy">
    <xs:sequence maxOccurs="1">
      <xs:element name="services" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:extension>
</xs:complexType>

<xs:element name="serviceData" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
<xs:element name="serviceObjectGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="unbounded" />
<xs:element name="advancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0" maxOccurs="1" />
<xs:sequence maxOccurs="1" />
</xs:complexType>

---

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:simpleType>
  <xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
  <xs:element name="mappedInterfaceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
  <xs:element name="original" type="NetworkOrIPRef" minOccurs="1" maxOccurs="1"/>
  <xs:element name="translated" type="NetworkObjectRefs" minOccurs="1" maxOccurs="1"/>
  <xs:element name="policyNAT" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
</xs:complexType>

<xs:complexType maxOccurs="1">
  <xs:element name="destAddress" type="NetworkObjectsRefs" minOccurs="1">
    <xs:complexType>
      <xs:element name="services" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:element name="serviceData" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:element>
</xs:complexType>

<xs:complexType maxOccurs="1">
  <xs:element name="protocol" type="IPTransportProtocol" minOccurs="1" maxOccurs="1"/>
  <xs:element name="originalPort" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
  <xs:element name="translatedPort" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
  <xs:element name="advancedOptions" type="FirewallNATAdvancedOptions" minOccurs="0"/>
</xs:complexType>

<xs:sequence maxOccurs="1">
  <xs:complexType name="PatOptions">
    <xs:element name="patAddressPool" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:choice>
          <xs:element name="patPoolAddressGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:complexType>
</xs:sequence>

<xs:complexType maxOccurs="1">
  <xs:element name="interfaceKeyword" type="xs:string" fixed="interface" minOccurs="0" maxOccurs="1"/>
</xs:complexType>

<xs:element name="isPatAllocatedInRoundRobin" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
<xs:complexType name="InterfaceNATManualFirewallPolicy">
  <xs:complexContent>
    <xs:restriction base="BasePolicy">
      <xs:element name="isRuleEnabled" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="section" minOccurs="1" maxOccurs="1"/>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">0</xs:restriction>
      </xs:complexType>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:element name="realInterfaceGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
<xs:element name="realInterfaceName" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="mappedInterface" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="mappedInterfaceGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="source" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:complexType>
<xs:element name="natType" type="NATType" minOccurs="1" maxOccurs="1"/>
<xs:element name="originalObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:element name="translated" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="objectGID" type="NetworkObjectRefs" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="destination" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="natType" type="NATType" fixed="Static" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:element name="originalObject" type="NetworkObjectRefs" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="translatedObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:element name="service" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:element name="originalObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:element name="transObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
</xs:complexType>
<xs:element name="isTransDNSReplies" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
<xs:element name="direction" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
<xs:restriction base="xs:string">
<xs:enumeration value="Unidirectional"/>
<xs:enumeration value="Bidirectional"/>
</xs:restriction>

Cisco Security Manager 4.4 API Specification (Version 1.1)

OL- 29074-01
<xs:simpleType>
<xs:element name="isNoProxyARP" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
<xs:element name="isRouteLookUp" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
</xs:element>
</xs:simpleType>

<xs:complexType>
<xs:element name="interfaceName" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="mappedInterface" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="originalObjectGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
<xs:element name="translated" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:complexType>

<xs:sequence>
<xs:element name="includeEmptyGroups" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="InterfaceNATObjectFirewallPolicy">
<xs:complexContent>
<xs:complexType base="BasePolicy">
<xs:extension name="section" fixed="2" minOccurs="1" maxOccurs="1">
<xs:simpleType>
<xs:restriction base="xs:unsignedInt">
<xs:enumeration value="1"/>
<xs:enumeration value="2"/>
</xs:restriction>
</xs:simpleType>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:sequence>
<xs:element name="name" type="PatOptions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:element>
<xs:complexType>
<xs:element name="isTransDNSReplies" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
<xs:element name="isNoProxyARP" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
<xs:element name="isRouteLookUp" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
<xs:element name="service" minOccurs="0" maxOccurs="1"/>
</xs:complexType>
</xs:element>

<xs:sequence>
<xs:element name="protocol" type="IPTransportProtocol" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:element name="originalPort" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
</xs:element>
</xs:complexType>

<xs:element>
<xs:complexType>
<xs:sequence>
<xs:element name="configureServiceMethods" type="IncludeEmptyGroups" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:complexType>

<xs:complexType name="GroupListRequest">
<xs:complexContent>
<xs:complexType base="BaseReqResp">
<xs:sequence minOccurs="1" maxOccurs="1">
<xs:element name="includeEmptyGroups" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
</xs:complexContent>
</xs:complexType>

</xs:element>
</xs:complexType>
</xs:sequence>
</xs:complexType>

Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:complexType name="InterfaceNAT64ManualFirewallPolicy">
  <xs:complexContent base="InterfaceNAT64ManualFirewallPolicy">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="isInterfaceIpv6" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
      <xs:element name="isNetToNet" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="GroupListResponse">
  <xs:complexContent base="BaseReqResp">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="deviceGroup" type="DeviceGroup"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceListByGroupRequest">
  <xs:complexContent base="BaseReqResp">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="deviceId" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="deviceCapability" type="DeviceCapability" minOccurs="1" maxOccurs="1"/>
      <xs:element name="deviceName" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="ipv4Address" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="gid" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InterfaceNATObjectFirewallPolicy">
  <xs:complexContent base="InterfaceNATObjectFirewallPolicy">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="isInterfaceIpv6" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
      <xs:element name="isNetToNet" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceListByCapabilityRequest">
  <xs:complexContent base="BaseReqResp">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="deviceCapability" type="DeviceCapability" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DeviceListByGroupRequest">
  <xs:complexContent base="BaseReqResp">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="deviceGroup" type="DeviceGroup"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
Cisco Security Manager 4.4 API Specification (Version 1.1)
<xs:element name="policyConfigDeviceResponse" type="PolicyConfigDeviceResponse"/>
</xs:complexContent>
</xs:complexType>
</xs:extension>
</xs:complexType>
</xs:complexContent>
</xs:complexType>
</xs:element name="policyObject" minOccurs="1" maxOccurs="1">
</xs:complexType>
</xs:element>
<xs:sequence minOccurs="1" maxOccurs="1">
<!-- ALL DEFINED POLICY OBJECT TYPES -->
  <xs:element name="networkPolicyObject" type="NetworkPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="portListPolicyObject" type="PortListPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="servicePolicyObject" type="ServicePolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="interfaceRolePolicyObject" type="InterfaceRolePolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="timeRangePolicyObject" type="TimeRangePolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="slaMonitorPolicyObject" type="SLAMonitorPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="aclPolicyObject" type="ACLPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="stdAcePolicyObject" type="StandardACEPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="extendedACEPolicyObject" type="ExtendedACEPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="identityUserGroupPolicyObject" type="IdentityUserGroupPolicyObject" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="securityGroupPolicyObject" type="SecurityGroupPolicyObject" minOccurs="0" maxOccurs="unbounded"/>

<!-- .... all other policy objects ... -->
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>
10.3 Event XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:csm="csm" targetNamespace="csm">
  <xs:include schemaLocation="common.xsd"/>
  <xs:simpleType name="EventType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="syslog"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="EventFormat">
    <xs:restriction base="xs:token">
      <xs:enumeration value="xml"/>
      <xs:enumeration value="plainText"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="EventCategory">
    <xs:restriction base="xs:token">
      <xs:enumeration value="configChange"/>
      <xs:enumeration value="deviceStatus"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="UpdateType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="NO_OOB"/>
      <xs:enumeration value="OOB"/>
      <xs:enumeration value="DEVICE_DOWN"/>
      <xs:enumeration value="DEVICE_UP"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="DeploymentType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Device"/>
      <xs:enumeration value="File"/>
      <xs:enumeration value="AUS"/>
      <xs:enumeration value="CNS"/>
      <xs:enumeration value="TMS"/>
      <xs:enumeration value="Unknown"/>
      <xs:enumeration value="NOT_APPLICABLE"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="BaseEventDetails">
    <xs:sequence>
      <xs:element name="subscriptionId" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="eventType" type="EventType" minOccurs="1" maxOccurs="1"/>
      <xs:element name="eventCategory" type="EventCategory" minOccurs="1" maxOccurs="1"/>
      <xs:element name="time" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
      <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:simpleType>
  <xs:simpleType name="DeviceSpecificEvent">
    <xs:complexType name="DeviceSpecificEvent">
      <xs:complexContent>
        <xs:restriction base="BaseEventDetails">
          <xs:element name="srcIP" type="xs:string" minOccurs="0" maxOccurs="1"/>
          <xs:element name="srcGID" type="ObjectIdentifier" minOccurs="0" maxOccurs="1"/>
          <xs:element name="srcDns" type="xs:string" minOccurs="0" maxOccurs="1"/>
          <xs:element name="srcOSType" type="OSType" minOccurs="0" maxOccurs="1"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:simpleType>
</xs:schema>
```

Cisco Security Manager 4.4 API Specification (Version 1.1)
10.4 Utility XSD

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2011 (http://www.altova.com) by BRIAN MCMACHON (CISCO) -->
<x:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema" xsi:schemaLocation="common.xsd">
  <x:simpleType name="Result">
    <x:restriction base="xs:token">
      <x:enumeration value="ok"/>
      <x:enumeration value="timeout"/>
      <x:enumeration value="failed"/>
    </x:restriction>
  </x:simpleType>
  <x:simpleType name="DeviceReadOnlyCLICmd">
    <x:sequence>
      <x:choice minOccurs="1" maxOccurs="1">
        <x:element name="deviceIP" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <x:element name="deviceName" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <x:element name="deviceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
      </x:choice>
      <x:element name="cmd" minOccurs="1" maxOccurs="1"/>
    </x:sequence>
  </x:simpleType>
  <x:simpleType name="DeviceCmdResult">
    <x:sequence>
      <x:element name="deviceIP" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <x:element name="deviceGID" type="ObjectIdentifier" minOccurs="1" maxOccurs="1"/>
      <x:element name="result" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <x:element name="resultContent" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </x:sequence>
  </x:simpleType>
  <x:simpleType name="ExecDeviceReadOnlyCLICmdsRequest">
    <x:extension base="BaseReqResp">
      <x:sequence>
        <x:element name="deviceReadOnlyCLICommand" type="DeviceReadOnlyCLICmd" minOccurs="1" maxOccurs="1"/>
      </x:sequence>
    </x:extension>
  </x:simpleType>
  <x:simpleType name="ExecDeviceReadOnlyCLICmdsResponse">
    <x:extension base="BaseReqResp">
      <x:sequence>
        <x:element name="deviceCmdResult" type="DeviceCmdResult" minOccurs="1" maxOccurs="1"/>
      </x:sequence>
    </x:extension>
  </x:simpleType>
</x:schema>
End of Document