Cisco Configuration Engine Software Development Kit API Reference and Programmer Guide 3.5.3

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

Text Part Number: OL-17661-04

Cisco Configuration Engine Software Development Kit API Reference and Programmer Guide 3.5.3

CONTENTS

About This Guide

Objective

The Cisco Configuration Engine Software Development Kit is a foundation technology for linking users to network services. Cisco Configuration Engine SDK accomplishes this by making applications network-aware and increasing the intelligence of the network elements.

The objective of *Cisco Configuration Engine Software Development Kit API Reference and Programmer Guide 3.5* is to show how to use the components included in the Cisco Configuration Engine SDK to write provisioning applications and access the Cisco Configuration Engine through the provided Application Programming Interface (API). Cisco Configuration Engine SDK includes APIs to allow:

- A node on the Configuration Engine Event bus to communicate with agent-enabled network devices, the Configuration Engine Event Gateway, and the Intelligent Modular Gateway (IMGW) via the Configuration Engine Event API. This allows you to write an application to configure one or more network devices.
- Access via a Java class to provision the IMGW Device Information Database that stores the Telnet user names, passwords, IP addresses of the devices and, additionally, enables adding, modifying and deleting devices from the data base.

This programmer guide provides general information about how the APIs and their functions can be used in your client application. Specific API details are in the API Reference chapters.

Who Should Use This Book

This guide is intended primarily for system integrators and application developers who need to develop applications that interact with the Configuration Engine and Configuration Engine Agents in network devices.

It is assumed that you have a basic understanding of network design, operation, and terminology, as well as familiarity with your own network configurations.

Related Documentation

Table 1 describes the related documentation available for Cisco Configuration Engine.

Table 1 Cisco Configuration Engine Documentation

Document Title	Available Formats
Cisco Configuration Engine Installation and Configuration Guide 3.5.3	This guide is available on Cisco.com.
Cisco Configuration Engine Administration Guide 3.5.3	This guide is available on Cisco.com.
Cisco Configuration Engine Software Development Kit API Reference and Programmer Guide 3.5.3	This guide is available on Cisco.com.
Troubleshooting Guide for Cisco Configuration Engine 3.5.3	This guide is available on Cisco.com.
Release Notes for Cisco Configuration Engine 3.5.3	This release notes is available on Cisco.com.

Conventions

When programming elements are cited in body text, the **bold** font is used for data types and objects, as well as functions and methods, and the *italic* font is used for variable names, as in the following:

- The class **Example** contains the function **do_this()** which takes one argument, *sum* of type **int.**
- The *italic* font is also used for file and directory names, for titles of books, and to introduce new words or terms.

Code examples are presented in the courier typeface, without bold or italic font:

```
class Example
{
   public:
      void do_this(int sum);
};
```

Cisco Developer Support Program

Cisco has a new support program for developers who are enabling products with Cisco supported interfaces. The Developer Support Program is being developed to provide formalized support for Cisco interfaces to accelerate the delivery of compatible solutions to Cisco customers.

The Developer Support Program offers the following benefits:

- Minimal support fees
- Flexible support model - purchase support in prepaid packs or for a specific period of time
- Consistent level of support - defined problem priority and escalation guidelines
- Deliver products to market faster – dedicated program with interface experts to assist you

The Developer Support Agreement, with the list of supported interfaces, is available on the Developer Support Web site at http://www.cisco.com/go/developersupport. Upon receipt of the signed agreement, a contract ID number and instructions will be generated for opening support cases with Cisco Developer Support Engineers.

For answers to your questions, you can contact Cisco Developer Support by e-mail at: developer-support@cisco.com.

CHAPTER 1

Product Overview

The Cisco Configuration Engine SDK offers a programmatic interface to the Cisco Configuration Engine 3.5.3. It allows users to write flow-through provisioning applications using the Cisco Configuration Engine (see Figure 1-1). The exposed interfaces are:

Configuration Engine Event API —(for details of the interface see Tibco documentation) The Configuration Engine Event Application Programming Interface (API) facilitates event-based communication between the user application, the Cisco Configuration Engine, and CNS Agents in network devices.

Note The existing applications using the event service and IMGW API should be recompiled with the 3.5.3 SDK to migrate to Cisco Configuration Engine 3.5.3.

Configuration Engine Namespace Mapper—The Namespace Mapper API, see Chapter 4, "Namespace Mapping Service," allows the user to map subjects in the Cisco Namespace to the application's namespace. It also provides the ability to configure a group of devices using one event.

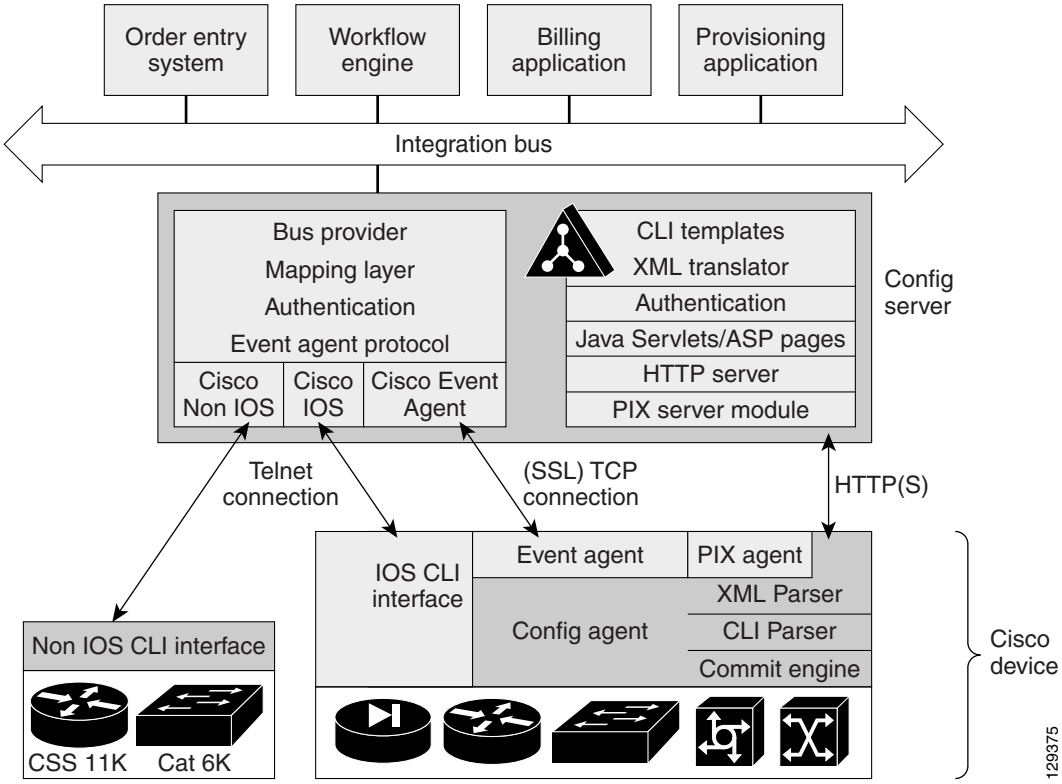
Template Management Interface—Using the Template Management Interface, see Chapter 3, "Configuration Service," users can create, delete or modify device templates stored in the host data store.

Admin, Config, Exec, Image Web Services API—The Web Services API, (see Chapter 5, "Web Services: Admin, Config, Image, Exec, NSM") provides administrative interface and operational interfaces in Web Services.

Group Admin, Namespace, Notification API—The Notification API is provided for the administration of group and namespace. The notification API allows application to get notification for group change and namespace change events.

Device Administration API— The Administration API allows administration of device such as device creation, modification, and deletion.

Figure 1-1 Configuration Engine Configuration Engine Components



Operating Modes

There are two modes in which the Cisco Configuration Engine can operate, the Internal Directory Mode and the External Directory Mode. Users who do not have a directory of their own can use the Cisco Configuration Engine in Internal Directory Mode. Users who have a directory of their own and wish to have more control of data, can use the External Directory Mode, in which the data store for the Cisco Configuration Engine will be the user-specified Lightweight Directory Access Protocol (LDAP) service.

Administration and Setup

Before using the Cisco Configuration Engine SDK to configure devices, the user must create corresponding device objects in the Cisco Configuration Engine data store, which could be an internal or external directory. The LDAP directory is used in the following circumstances (and if any of these are true, the device objects must be created in the LDAP directory store, using Config Engine GUI, or the bulk upload facility):

- If the Configuration Engine event agent in the device connects to the event gateway using encrypted connections
- If the Configuration Engine config agent accesses the Configuration Engine Config Server using HTTP or HTTPS

Administrative Steps for Internal Directory Mode

After Cisco Configuration Engine is set up in the Internal Directory Mode, the user can use the Cisco Configuration Engine GUI to populate the data store. The schema for the devices is pre-defined, although device attributes can be added by using the Cisco Configuration Engine GUI. The directory containment hierarchy is also fixed.

Administrative Steps for External Directory Mode

Loading the Device Schema

The process of designing and loading the device schema is independent of configuring the system using the setup program described in the installation and configuration guides.

Step 1 Verify that the pre-existing external directory is running. Verify by using utilities and tools supplied by the directory vendor.

Step 2 Design the device schema. A schema example is provided in Appendix B, "Sample Schema." In addition, you need:

- The unique ConfigID of the device
- The unique EventID of the device
- The configuration template associated with the device Optional multi-valued attribute of type 'Distinguished Name' that refers to the groups to which the device belongs. A multi-valued attribute to store the list of Groups to which the device belongs.

Step 3 Load the device schema designed in the previous step, using directory vendor supplied tools. Then, load the NSM schema for group, application, and event objects.

Step 4 Populate objects in the directory using Cisco Configuration Engine.

Step 5 Verify system operation by logging into "config-server" (refer to the *Cisco Configuration Engine Administration Guide*).

Step 6 Verify that host system is running. Check by trying to log in. Refer to the *Cisco Configuration Engine Administration Guide* for details. Configure the host system using the **Setup** program described in the *Cisco Configuration Engine Installation and Configuration Guide 3.5*.3.

Step 7 Check for network connection between host system and the directory server. Verify by "pinging" from the host system using the server IP address or the hostname. If everything is OK, you are ready to start using the system.

SDK Directories

The CE-SDK is installed in the following subdirectories of /*<INSTALLDIR>/CSCOesdk* for Solaris:

- bin – Event service daemons and libraries
- conf – configuration and properties files
- docs – Event service documentation
- include – C++ header files
- lib – library files
- java – .jar files for SDK
- sample – sample application files
- schema – NSM schema directory for iplanet and NDS
- src – event service samples src
- templates – SDK internal system files
- tools – cns-listen, cns-send, and web service test tool
- xml – IOS Agents DTD and XSD files

CHAPTER 2

Event Service and IMGW

Cisco customers, such as service providers and enterprise IT managers, want the ability to tailor their network services to meet user and application demands. For example, service providers want to be able to offer personalized levels of Quality of Service to their customers. Enterprise IT managers want the ability to centrally define and automate network policies. To accomplish these goals, network devices must be able to generate (produce) and respond to (consume) events. This is the purpose of Event Service.

Event Service is based on TIBCO TIB/Rendezvous version 7.2 software. This release is focused on enabling Cisco network elements to communicate with TIB/Rendezvous applications by exchanging messages.

TIB/Rendezvous software consists of two interacting components, the TIB/Rendezvous Application Programming Interface (API) library and the TIB/Rendezvous daemon. Applications call API functions to produce and subscribe to events. The TIB/Rendezvous daemon is the communications process that ensures the reliable message transmission. Application processes and the TIB/Rendezvous daemon do not have to reside on the same machine.

With Cisco Configuration Engine, Intelligent Modular Gateway (IMGW) supports many device types including support for legacy devices that do not have the embedded software agents. The purpose of IMGW is to enable communication between applications on the Tibco event bus (the configuration server provides template-based services that allow a network administrator to manage a large number of routers on a network) and legacy devices in the network.

Event Service Overview

Event Service consists of three components:

- Event Service API on Solaris version 2.10.

  Cisco has licensed the TIB/Rendezvous software from TIBCO. The TIB/Rendezvous API library and the TIB/Rendezvous daemon are included in the Software Development Kit (SDK). This allows developers to write event-driven applications to communicate with Cisco applications using TIB/Rendezvous software.

- Event Gateway on the Cisco Configuration Engine.

  Cisco Event Gateway enables network elements to publish and subscribe to events. This allows developers to write event-driven applications to communicate with Cisco network elements. Also, the Event Gateway acts as an interface to the Event bus.

- Cisco IOS Event Agent

Cisco IOS Event agent communicates with Event Gateway. It allows an application running on Cisco IOS to publish and subscribe to events.

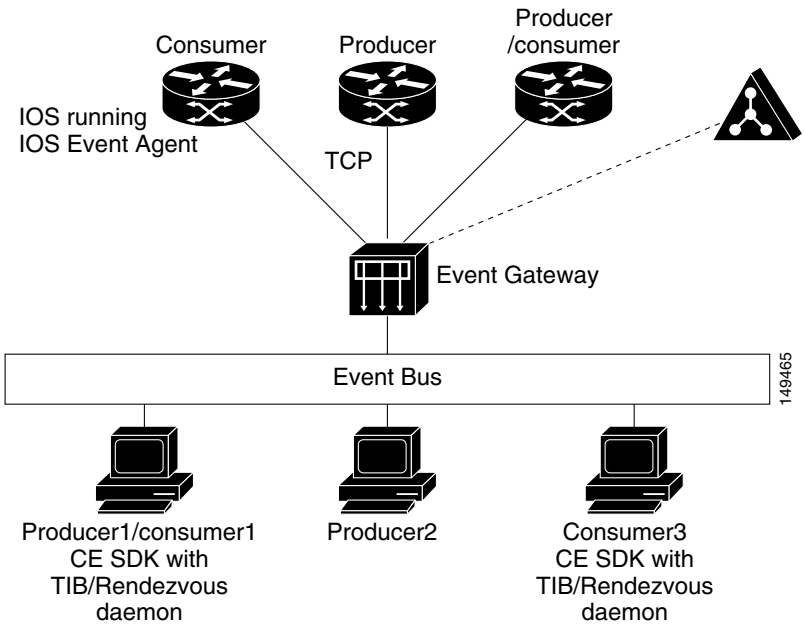
Cisco IOS is not included in the Configuration Engine product. You need Cisco IOS release 12.2(2)T or later to use Event Agent.

The TIB/Rendezvous software release 7.2 documentation is included with Cisco Configuration Engine SDK. The documentation set includes:

- TIB/Rendezvous Concepts
- TIB/Rendezvous Administration
- TIB/Rendezvous C++ Reference
- TIB/Rendezvous Java Reference

Figure 2-1 shows the Event Service implementation.

Figure 2-1 Event Service



Producers can send messages on the TIB/Rendezvous network. Consumers can listen to messages on the TIB/Rendezvous network. The TIB/Rendezvous daemon process runs on every participating computer on the TIB/Rendezvous network. It is responsible for all the communications process. An Cisco IOS device sends and listens to messages through the use of an Event Gateway which uses the TIBCO API.

Note The TIBCO API supports a number of platforms. However, Cisco Configuration Engine SDK only supports Solaris version 2.10.

Note The existing applications using the event service and IMGW API should be recompiled with the 3.5.3 SDK to migrate to Cisco Configuration Engine 3.5.3.

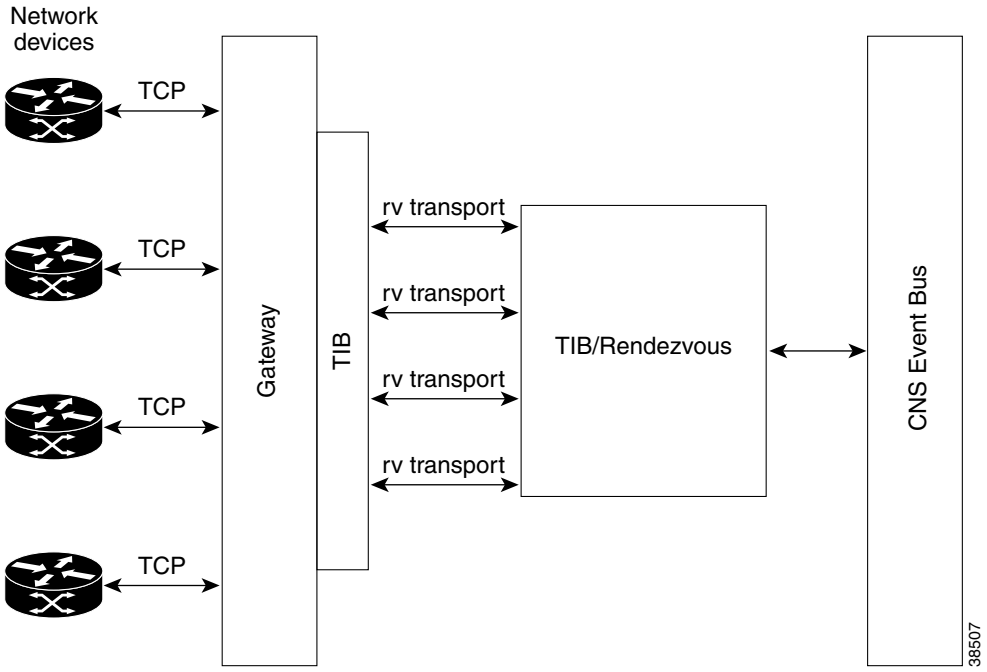
Each Cisco IOS device maintains a permanent Transmission Control Protocol (TCP) connection with the Event Gateway. The Event Gateway and the TIB/Rendezvous daemon only reside on the host system. Every event message passes through the TIB/Rendezvous daemon. The TIB/Rendezvous daemon dispatches messages to subscribers, posts messages to the TIB/Rendezvous network, filters subject-addressed messages, and guarantees packet delivery.

As shown in Figure 2-1, a device can either be a producer, a consumer, or both. Producer and consumer applications can be running on IOS, Solaris version 2.10.

Event Services Structure

The network devices open one TCP session per device to the Event Gateway as shown in Figure 2-2. For every session on the TCP pipe, the gateway opens a Rendezvous transport to the TIB/Rendezvous daemon through the TIBCO API interface. Requests to register and unregister consumers, requests to post events, and event notification happen through the gateway, which acts as an interface between the devices and the Rendezvous daemon.

Figure 2-2 Event Services



The TIB/Rendezvous daemon is responsible for passing messages between the Rendezvous transports. The Rendezvous routing daemon (**rvrd**) is used for routing messages across subnets.

When a TCP session is severed, the gateway terminates the corresponding Rendezvous transport. The termination of the transport leads to the clearing of resources allocated for that transport in the daemon. An advisory message will then be sent to notify that the terminated transport is no longer valid.

The termination of a transport might lead to a subscription path being removed, if that transport was the only listener for a particular subject. This prevents a message with that subject name from being placed on the event bus.

Event Service Features

The Event Service supports the following features:

- Detection of lost sessions, page 2-4
- Reliable delivery of messages, page 2-4
- Subject-based addressing, page 2-4
- Advisory messages, page 2-4
- Deployment configuration, page 2-4

Detection of lost sessions

If the connection between any network device and the Event Gateway gets severed, the gateway detects it and closes any sessions that are open for that client and frees up any resources allocated for that session.

Reliable delivery of messages

The Event Service uses reliable delivery of messages which retains outbound messages for 60 seconds. For information on reliable delivery of messages, refer to the TIB/Rendezvous Concepts manual. Event Service does not use certified messaging.

Subject-based addressing

Every message transmitted carries a subject name, that specifies the destination(s) of the message and can also describe the message content. Subject-based addressing technology helps eliminate details of network addresses, protocols and so on. Subject-based addressing technology provides a uniform name space for messages and their destinations. Also, communication between applications is anonymous, consumers need not know where or how data is produced, and producers need not know where data is consumed, or how it is used. They only need to agree to label data items with the same set of names, and that the actual data be in form that both can manipulate and interpret.

Advisory messages

The TIB/Rendezvous software sends messages to indicate errors, warnings and other information pertaining to the state of the network, the state of the daemon or the state of the transports connected to a daemon. For example, when a Rendezvous transport is terminated, the Rendezvous daemon sends out an advisory message notifying any listeners that the particular transport is no longer available.

Deployment configuration

The host system with the Event Gateway and producer/consumer applications can reside on different subnets. If they are on different subnets, the Rendezvous routing daemon must be configured. Refer to the TIB/Rendezvous Administration manual for configuration information.

Intelligent Modular Gateway

The Intelligent Modular Gateway module provides the means to access devices that do not have embedded event agents by simulating results of such events via a telnet session to the device. Customers will be able to run the existing event gateway for managing event agent-enabled devices and, simultaneously, run the IMGW to manage devices that lack the event agent.

The IMGW will offer several interfaces to the user: as a node on the Tibco event bus, as an event gateway on the router network, and as a servlet engine. The first will enable applications such as Configuration Server to communicate with the IMGW via the Event Bus. The second interface will serve as the event gateway on the router network enabling the routers to receive their configurations from the IMGW. The third will be used to handle entry of username, password and IP address data for IMGW. The IMGW feature list is shown in the table below.

Table 2-1 Principal features of the Intelligent Modular Gateway appliance

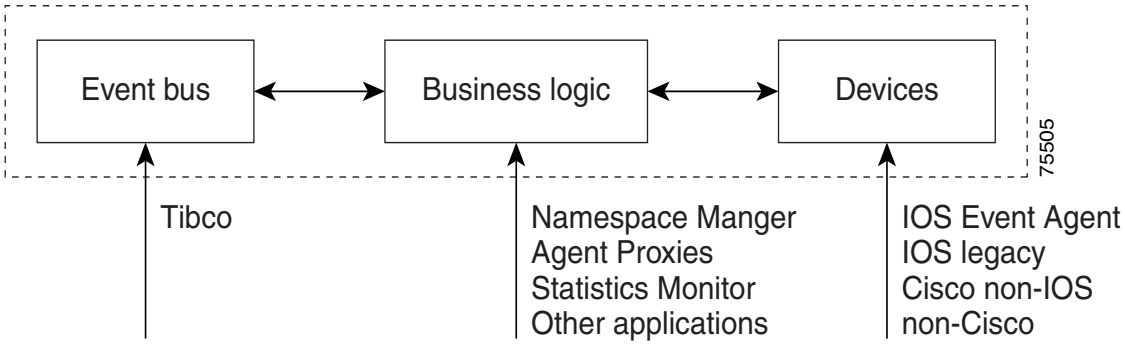
No.	Feature	Comments
1	Standard Telnet Interface support	Supports devices that are RFC 854 compliant
2	Support for non-Cisco IOS and legacy Cisco IOS devices	Supports CatOS (release 6.1) and legacy (release 12.0) devices
3	Support for CSS 11k Catalyst 6k/4k, PIX Firewall and Content Engine	Yes
4	Operates concurrently with the Event Gateway	Yes, but the data will not be shared and namespace mapper groupings must be configured separately for each gateway
5	Acts as proxy for Cisco IOS agents	Yes
6	Multiple end devices	Yes
7	Communicates via northbound interface	Yes, but requires provisioning applications to set up telnet login usernames, passwords and device IP addresses prior to using Config Agent.
8	Offers password security	Yes
9	Offers data security	Yes, provided the particular device supports SSH encryption (SSH authentication is not supported)
10	Status reporting	Status is provided via response events on the Event Bus
11	Multiplatform support	Linux and Solaris are supported at this time

The current Event Gateway works only with devices that have Event Agent functionality and performs three separate functions that include:

- Interfacing with the event bus
- Handling namespace mapping
- Communications with an IOS-based device

Internally, IMGW separates these three functions into separate logical units as shown in Figure 2-3 below. The intent behind this separation is to minimize dependencies between modules and reduces the risk that changes to one module will drastically affect the operation of another module. Furthermore, it enables changing each unit individually to upgrade its performance, as new applications grow to acquire new functionality.

Figure 2-3 IMGW separates Event Gateway functions into three separate modules



Event Bus Interface

The Event Bus interface handles all interactions with external workflow engines. You can subscribe to or publish events on this Event Bus. The current Event Gateway uses the Tibco Event Bus protocol. This logic is separated to enable using a different Event Bus, such as CORBA, or even use an entirely different event management scheme.

Device Interface

The device interface has been enhanced and modularized to allow support for many different types of devices. Currently, the Event Gateway only supports devices running the Cisco IOS version of the Event Agent. Because there are many different versions of Cisco IOS in use, it is desirable to support older versions of IOS. In addition, there are many non-Cisco IOS devices (such as CatOS-based switches) that are a major part of Cisco's product line. Such devices can be supported by a Telnet Gateway module. The advantage of this type of architecture is that at some point in the future, if desired, it is extensible to new types of interaction with other devices. For example, there are many non-Cisco devices on the market that do not have a **telnet** daemon in them, but rather, are controlled entirely by HTTP. With this type of architecture, an "HTTP Gateway" module could be written to communicate with such devices.

Event Bus Interface

To send a configuration to a device, a node on the Event Bus should publish on a subject of the format **cisco.mgmt.cns.config.load.dev_id** and possibly **cisco.mgmt.cns.config.sync-status.dev_id,** and subscribe to the **cisco.mgmt.cns.config.complete.dev_id**, **cisco.mgmt.cns.config.failure.dev_id**, and **cisco.mgmt.cns.config.warning.dev_id** subjects. The functions and formats of these messages are described earlier in this document. The **.dev_id** suffix is translated by the Namespace Mapper into two pieces of information: an internal subject without the suffix (for example, **cisco.mgmt.cns.config.load**) and a device ID (inferred from the suffix). This simplifies matters for application processing modules,

such as the Configuration Agent, because they do not need to know anything about decoding the destination of an event from the subject. It also allows router groups to be defined, where the suffix does not refer to an individual router, but rather, to a group of routers.

Note that it is not possible to do a two-stage commit when using telnet. The purpose of this event is to inform devices to apply a previously downloaded configuration, so as to minimize the time during which there is an inconsistent state of network element configuration. However, because the devices do not have intelligence in them to hold an un-applied configuration, it is not possible to implement this event using this gateway. Therefore, this event is ignored if it is sent by applications.

An example of XML payload for configuration is:

```
<config-event config-action="read" no-syntax-check="FALSE">
<identifier>IDENTIFIER</identifier>
<config-data>
    <config-id>AAA</config-id>
    <cli>hostname Jigglypuff</cli>
</config-data>
</config-event>
```

To distinguish failures to log in to the device from failures after login, negative numbers in the **line-number** field are used to indicate a failure to log in to the device. A **line-number** value of "-1" indicates a transport failure (there was either no route to the destination, an invalid IP address, or the device refused the telnet connection). A **line-number** value of "-2" indicates a login failure (the telnet connection was opened properly, however, the user ID or password supplied was invalid).

An example of an incorrect password error is:

```
<config-failure>
<identifier>OSIRIS-030500-1330</identifier>
<config-id>Router1-030500</config-id>
<error-info>
<line-number>-2</line-number>
<error-message>% Invalid username or password</error-message>
</error-info>
</config-failure>
```

It is possible for a successful event to generate output. The reason is that when processing configurations line-by-line, various warning messages can be output, and unlike the embedded agent, there is no way to access the internal parser return codes to identify whether it was a success or a failure. Therefore, any output returned as a response to a configuration command is returned to the caller. While the TGServer program identifies certain special messages as errors (especially those involving syntax), there are many random messages that are output by Cisco IOS that cannot easily be determined to be a success or failure. Such messages are returned so that applications can make the necessary decisions. In this case, a **config-warning** tag is returned on the **cisco.mgmt.cns.config.warning.dev_id** event. This should be interpreted as a success, however, the **error-info** value should be logged so that if, in fact, an error did occur, it can be investigated.

Following is an example:

```
<config-warning>
<identifier>OSIRIS-030500-1330</identifier>
<config-id>Router1-030500</config-id>
<warning-message>**CLI line # 0: New switch type will take effect upon
reload.</warning-message>
</config-warning>
```

To receive a configuration from a router, a node on the Event Bus should publish on the **cisco.mgmt.cns.exec.cmd.dev_id** subject and should listen on the **cisco.mgmt.cns.exec.rsp.dev_id** subject.

A brief description of the relevant parts of the Document Type Descriptions (DTDs) for the data contained in these messages follows below. The XML DTD for the data for the input **exec** command request that is sent out on the **cisco.mgmt.cns.exec.cmd.dev_id** event is as follows:

```
<!ELEMENT exec-cmd-event (identifier-exec, (event-response|server-response)?,
cli-exec)>
<!ELEMENT identifier-exec (#PCDATA)>
<!ELEMENT event-response (reply-subject)>
<!ELEMENT reply-subject (#PCDATA)>
<!ELEMENT server-response (ip-address-exec, port-number?, url)>
<!ELEMENT ip-address-exec (#PCDATA)>
<!ELEMENT port-number (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT cli-exec (#PCDATA)>
```

While the generic **exec-cmd-event** element supports numerous modes of execution, including a customizable **reply-subject** and multiple methods of response. Following is an example of a request to read the configuration:

```
<exec-cmd-event>
<identifier-exec>ID_0001</identifier_exec>
<cli-exec>show running-config</cli-exec>
</exec-cmd-event>
```

The **exec-cmd-event** element contains the element identifier which is used to correlate the output event with the input event. The **cli-exec** element contains a single **exec** CLI command line.

The XML DTD for the data for the output of the input **exec** CLI command request that is sent out on the **cisco.mgmt.cns.exec.rsp.dev_id** event is as follows:

```
<!ELEMENT exec-cmd-response (identifier-exec, status, (error-info|response))>
<!ELEMENT identifier-exec (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT response (#PCDATA)>
```

The **identifier-exec** element is used to correlate the output event with the input event. The **status** element indicates whether the CLI **exec** command failed or succeeded. The value of the **status** element is set to *fail* or *success*. The **response** element contains the output from the **exec** CLI command. Note that the **status** element may contain the value *success* if a semantic, rather than a syntax, error occurred. Therefore, applications should always check the value of the **response** element to verify that it is in the expected format. The **error-info** element is in the same format as described earlier for the **config-warning** and **config-failure** tags.

Following is an example of a **response** element:

```
<exec-cmd-response>
    <identifier-exec>ID_0001</identifier-exec>
    <status>success</status>
    <response>

Building configuration...

Current configuration : 985 bytes
!
version 12.2
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname r1
!
logging rate-limit console 10 except errors
```

```
end

    </response>
</exec-cmd-response>
```

IMGW Device Information Database

The IMGW Device Information Database, or the Data Store, stores the information that is required to log in, through telnet, to the individual devices. Provisioning applications must store such information in this data store after acquiring it from the user, in this case, an administrator. The API that is provided for accessing the Data Store, is described in the following sections.

You can use the Cisco Configuration Engine GUI that enables you to populate and manage the data in the directories. More detailed information about how to use the GUI is available in the *Cisco Configuration Engine Administration Guide*.

Data Structures

The **HopInfo** structure is a generic method for specifying how to log in to a device. Because of the many layers of authentication in many network devices, it is necessary to have an arbitrary number of usernames and passwords to present to the device. Furthermore, sometimes devices are not directly connected to the management network and require a proxy device to log in. The **HopInfo** structure is designed with these needs in mind.

Examples of uses of this structure include:

- Devices with a secondary *privileged* mode (such as Cisco IOS enable mode)
- Additional authentication modes such as TACACS
- Embedded-within-embedded applications, such as line cards on a Catalyst switch, MGX, or a 15454 optical concentrator.

These *composite* devices have a login for the central operating system and a command to access the independent operating system running on a line card. Both privileged mode and line card access require a login, however, not a hop to a different device. Therefore, in the following discussion, they are referred to as *virtual* hops.

The fields in the **HopInfo** structure are shown in Table 2-2.

Table 2-2 HopInfo Structure

Field	Purpose
device_type	The type of device. Also used as the base name for scripts. (string)
ip_address	IP address of the device. (string)
port	TCP port on which to access the device. (integer)
username	Username with which to log in to the device. (string)
password	Password with which to log in to the device. (string)

The **device_type** field indicates the script to be used to execute the actual commands for the device. A file maps the various device types to application and script names. Initially, all supported device types will execute the application expect script, however, the name of the script is different for each device type.

The table of **HopInfo** records is interpreted as follows. If an IP address exists in the record, it is necessary to telnet into a new device at that point. If an IP address does not exist in the record, it is considered to be a virtual hop, that is, an embedded line card is being accessed or an additional authentication mode of the current device is necessary. These records can be chained to accomplish any combination of logins.

The **port** field identifies which port to use. This is especially useful when dealing with communication servers (such as a 2511) because they reserve the port range 2001-2016 to access the sixteen outgoing async lines. If the **port** field is not present, it defaults to 23 to allow a normal telnet (except in the case of a Secure Shell (SSH) login where the default port is 22, as shown in Table 2-4).

Examples

Supported Device Types

The following tables describe the *Hopinfo* structure for Devices that are directly accessible on the network by IMGW. For Access via Commserver please refer to Table 2-11, on page 11.

The fields marked with "X" are mandatory. Also, all the rows in *Hopinfo* Structure are mandatory even though they may be null. The exception is the *device_type* field, which may not be null.

While accessing the device through Commserver, port username must be null, because port username is not supported.

Table 2-3 Cisco IOS Device Directly Connected

device_type	ip_address	port	username	password
IOS_LOGIN	X		X	X
IOS_EN			X	X

Table 2-4 Cisco IOS Device Directly Connected, supporting SSH

device_type	ip_address	port	username	password
IOS_LOGIN:SSH	X		X	X
IOS_EN			X	X

Table 2-5 Catalyst Device Directly Connected

device_type	ip_address	port	username	password
CATALYST_LOGIN	X		X	X
CATALYST_EN			X	X

Table 2-6 Catalyst IOS MSFC Blade Directly Connected

device_type	ip_address	port	username	password
CATALYST_LOGIN	X		X	X
IOS_CAT_BLADE		X	X	X
IOS_EN			X	X

Table 2-7 CatIOS Device Directly Connected

device_type	ip_address	port	username	password
CATIOS_LOGIN	X		X	X
CATIOS_EN			X	X

Table 2-8 CSS Device Directly Connected

device_type	ip_address	port	username	password
CSS_LOGIN	X		X	X
CSS_EN			X	X

Table 2-9 CE Device Directly Connected

device_type	ip_address	port	username	password
CE_LOGIN	X		X	X
CE_EN			X	X

Table 2-10 PIX Device Directly Connected

device_type	ip_address	port	username	password
PIX_LOGIN	X		X	X
PIX_EN			X	X

When any of the above devices are accessed via Commserver, the resultant hopinfo structure shall have the following two rows appended with the respective hopinfo for that device.

Table 2-11 Partial Hopinfo for Commserver Access

device_type	ip_address	port	username	password
COMMSERVER_LOGIN	X		X	X
COMMSERVER			X	X

Hopinfo Examples

Table 2-12 Cisco IOS Device Directly Connected

device_type	ip_address	port	username	password
IOS_LOGIN	172.28.6.90		johndoe	passnow
IOS_EN			enuser	encompass

Table 2-13 Cisco IOS Device Access Via SSH

device_type	ip_address	port	username	password
IOS_LOGIN:SSH	172.28.6.90		johndoe	passnow
IOS_EN			enuser	encompass

Table 2-14 Cisco IOS Device Connected Via Commserver

device_type	ip_address	port	username	password
COMMSERVER_LOGIN	172.28.6.90		janedoe	passlogin
COMMSERVER		2001		supercomm
IOS_LOGIN			johndoe	passnow
IOS_EN			enuser	encompass

Table 2-15 Catalyst IOS MSFC Blade Directly Connected

device_type	ip_address	port	username	password
CATALYST_LOGIN	172.28.6.90		catdoe	catpass
IOS_CAT_BLADE		15	bladoe	sharp
IOS_EN			enuser	encompass

Table 2-16 Catalyst IOS MSFC Blade Via Commserver

device_type	ip_address	port	username	password
COMMSERVER_LOGIN	172.28.6.90		janedoe	passlogin
COMMSERVER		2001		supercomm
CATALYST_LOGIN	172.28.6.90		catdoe	catpass
IOS_CAT_BLADE		15	bladoe	sharp
IOS_EN			enuser	encompass

Depreciated IMGW API

Note The IMGW API provided in IMGWDeviceClient.jar has been deprecated. These APIs do not work together with the new Device Admin API or the Admin Web Service APIs.

The API modules used to access the Device Information Database are shown in Table 2-17.

Table 2-17 API for Device Information Database

Function	Purpose
createDevice(<*dev_id*>, <*gateway_id*>)	Create a device. Throws a DeviceExists exception if it already exists. Causes the gateway with the ID <*gateway_id*> to automatically subscribe to events for this device on the Event Bus.
addDeviceHop(<*dev_id*>, <*obj*>)	Add a device hop record to the end of the table of **HopInfo** structures.
getDeviceHop(<*dev_id*>,< *n*>)	Get the data for the device hop <*n*>. <*n*> = 0 indicates the first row of the table and <*obj*> is returned. If the supplied <*n*> is past the end of the table for this device, NULL is returned.
deleteDevice(<*dev_id*>)	Delete a device from the Device Information Database. Throws a DeviceNotFound exception when the device does not exist. Causes the gateway to automatically unsubscribe from events for this device on the Event Bus.

In the above table, <*dev_id*> and <*gateway_id*> are strings, <*n*> is an integer, and <*obj*> is a structure of the type **HopInfo**, for which the format is described in Table 2-2 on page 2-9.

It should be noted that not all possible configurations of the **HopInfo** structure are supported initially. Specifically, the TGServer program only supports the following access methods:

- Direct connection to the device through telnet or SSH (that is, the device is directly accessible through a locally visible IP address). An example is shown in Table 2-13.
- Connection through a Cisco 2511 communication server to the console port of the device through reverse telnet. In this case, extra usernames and passwords for the communication server are supported, because the TGServer program handles this as a special case. Examples are shown in Table 2-14 and Table 2-16.
- Connection to a sub-device of a supported compound device. The TGServer program handles this as a special case for the particular device. In this case, Table 2-15 is supported as shown, but only if using a Catalyst 6000 series device with an Cisco IOS blade. Other compound devices, such as the MGX or the 15454, are not supported.

Simulated Agents

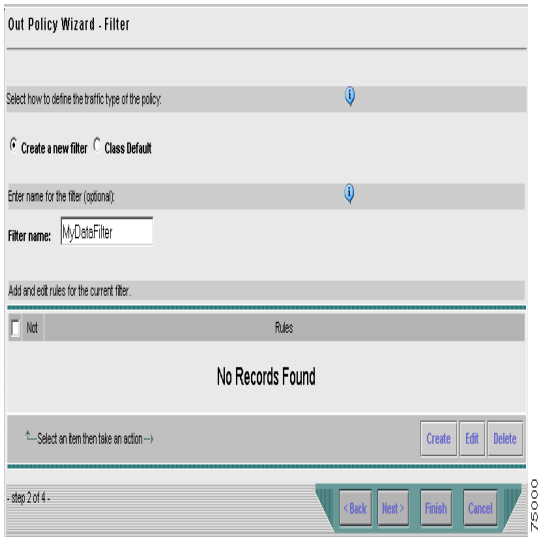
IMGW simulates both, the configuration agent and the image agent for all the devices in its repository.This poses a potential problem if the client wants the IMGW to simulate only one of theose agents, if they already have he other functionality achieved through the corresponding configuration

agent. For example, say the client wants to use the configuration agent on the device, but at the same time use the IMGW only for image upgrades. In this situation the device will be configured twice, once by the agent and once by the IMGW.

The protocol between the IMGW servlet and its clients has been modified to include a new element called **simulatedAgent**. Populating the Data Store

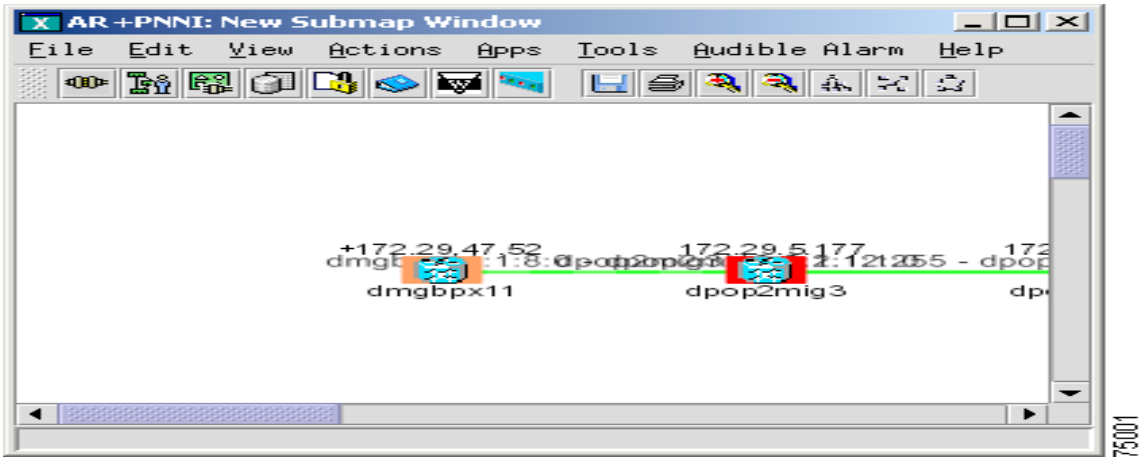
You can use the API as described in chapters 8, 9, and 10 of this document, or you can use these APIs to write your own GUI. The overall API diagram is shown in Figure 2-4.

Figure 2-4 Administrator's API for populating the Data Store



The Administrator populates the Data Store with device information by invoking the **createDevice()** method. A typical progression of calls is shown in the sequence chart in Figure 2-5. This call is needed to establish which devices will be serviced by which IMGW. If the call is successful, it is published to the Event Bus. Similarly, the **deleteDevice()** call, if successful, publishes the deleted device to the Event Bus by sending a *cisco.mgmt.cns.gateway.device.delete* message. The other APIs, **addHopInfo()**, **listDevices()**, and **getHopInfo()** have similar sequence diagrams but do not interface with the Event Bus.

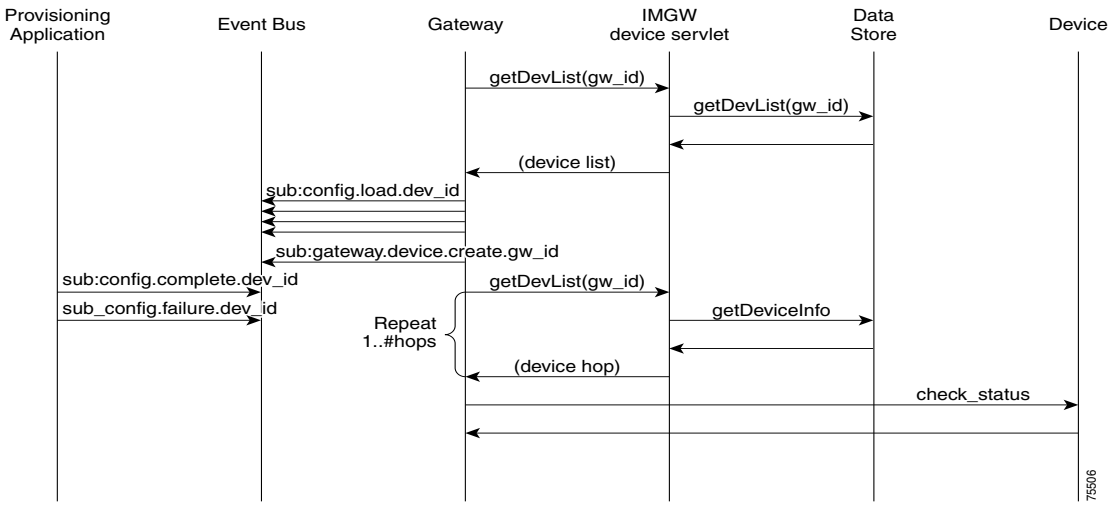
Figure 2-5 A typical sequence diagram for a create operation is shown. Note that the call must be repeated for every device being registered by the administrator.



End User Interface

There are two external interfaces to Config Engine: the Event Bus interface and the Device Information Database interface. The Event Bus interface is through the Tibco Event Bus APIs, and the Device Information Database interface is through a Java client library that communicates with a servlet. The servlet functions as a front-end for the actual data store, which is a directory. A sequence diagram, shown in Figure 2-6, identifies the action.

Figure 2-6 Event Bus Sequence Diagram



CHAPTER 3

Configuration Service

The Configuration Service is the core component of the Configuration Engine. It consists of a configuration server that works in conjunction with configuration agents located at each network device. The Configuration Service delivers device and service configurations to Cisco IOS devices for initial configuration and mass reconfiguration by logical groups. Routers receive their initial configuration from the Configuration Service when they start up on the network the first time.

The Configuration Service uses the Event Service to send and receive events required to apply configuration changes and send success and failure notifications.

The configuration server consists of a web server that uses configuration templates and the device-specific configuration information stored in the internal directory or external directory modes.

Configuration templates are text files containing static configuration information in the form of command-line interface (CLI) commands. In the templates, variables are specified using lightweight directory access protocol (LDAP) URLs that reference the device-specific configuration information stored in a directory.

The configuration server uses Hypertext Transport Protocol (HTTP) to communicate with the Configuration Agent running on the managed Cisco IOS device. The configuration server transfers data in eXtensible Markup Language (XML) format. The configuration agent in the router uses its own XML parser to interpret the configuration data and remove the XML tags from the received configuration.

The configuration agent can also perform a syntax check on received configuration files. The configuration agent can also publish events through the event gateway to indicate the success or failure of the syntax check.

The configuration agent can either apply configurations immediately or delay the application until receipt of a synchronization event from the configuration server.

Configuration Service Overview

Configuration Service consists of the following components:

- Cisco IOS Configuration Agent

  The Configuration Agent feature supports routing devices by providing the following:

  - Initial configurations
  - Incremental (partial) configurations
  - Synchronized configuration updates

  IOS is not included in the Configuration Engine product. You need IOS release 12.2(2)T or later to use Event Agent.

- Template File Manager is an external XML interface that can be used to integrate the Cisco Configuration Engine into existing provisioning applications.

Template File Manager

The Template File Manager provides an external XML interface that allows a client to manage template files on the Cisco Configuration Engine. The Java servlet performs template file management operations such as import/export/remove template. The operation type and the required parameters to perform the operation are specified through an XML payload request message submitted to the servlet via an HTTP POST.

Usage

The Template File Manager consists of a Java HTTP servlet that receives XML requests via an HTTP POST. The grammar for this XML request, and its subsequent XML response is defined in the Configuration Engine file management DTD. An XML request (as in the example Import Template "import.test.cfgtpl") must be submitted as form data using the MIME content type multipart/form-data, as the value of the parameter named *command* (i.e. a name/value pair where the name is *command* and the value is the XML content). For more information on multipart/form-data see RFC 1867 at http://sunsite.dk/RFC/rfc/rfc1867.html.

Configuration Engine File Management DTD

The Configuration Engine File Management DTD can be obtained directly from Cisco Configuration Engine by going to the following URL:

http://<Cisco Configuration Engine hostname>/config/cns_file_management.dtd

The following section describes the Configuration Engine XML DTD to support import/export/remove template file operations (request & response).

Common Configuration Engine DTD

```
<!ELEMENT cns-request ( message-id, security, message )>
<!ELEMENT cns-response ( message-id, status, message )>

<!ELEMENT message-id (#PCDATA)>
<!ELEMENT security (username, password)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!-- note: status can be "success" or "failure" only -->
<!ELEMENT status (#PCDATA)>
```

All Configuration Engine services inside message tag

```
<!ELEMENT message ( file-management )>
```

File Management service

```
<!ELEMENT file-management ( import-files | export-files | remove-files | list-files |
files-result  | list-files-result | export-files-result)>
```

Import-files tag

```
<!-- Interface: import-files -->
<!ELEMENT import-files (import-file+)>
<!ELEMENT import-file (file-id, source-url, target-file, overwrite, content?)>
```

Export-files tag

```
<!-- Interface: export-files -->
<!ELEMENT export-files (export-file+)>
<!ELEMENT export-file (file-id, source-file, target-url)>
```

Remove-files tag

```
<!-- Interface: remove-files -->
<!ELEMENT remove-files (remove-file+)>
<!ELEMENT remove-file (file-id, source-file)>
```

List-files tag

```
<!-- Interface: list-files -->
<!ELEMENT list-files (query)>
<!ELEMENT query (patterns)>
<!ELEMENT patterns (pattern+)>
<!ELEMENT pattern (filename, path)>
```

Export-files-result tag

```
<!-- Interface: files-result -->
<!ELEMENT files-result (result+)>
List-files-result tag (not supported in Cisco IE2100 Series Configuration Registrar
release 1.1)
<!-- Interface: list-files-result -->
<!ELEMENT list-files-result (list-result+)>
<!ELEMENT list-result (filename, path)>
<!-- Interface: export-files-result -->
<!ELEMENT export-files-result (export-result+)>
<!ELEMENT export-result (result, content?)>
```

Common tags

```
<!ELEMENT file-id (#PCDATA)>
<!ELEMENT source-file (filename, path)>
<!ELEMENT source-url (url, security?)>
<!ELEMENT target-file (filename, path)>
<!ELEMENT target-url (url, security?)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT filename (#PCDATA)>
<!ELEMENT path (#PCDATA)>
<!-- note: overwrite can be true or false only -->
<!ELEMENT overwrite (#PCDATA)>
<!ELEMENT content (#PCDATA)>

<!ELEMENT result (file-id, error-info)>
<!ELEMENT error-info (error-code, error-message)>
<!ELEMENT error-code (#PCDATA)>
<!ELEMENT error-message (#PCDATA)>
```

Template File XML Response Messages

Example 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<cns-response>
    <message-id>6</message-id>
    <status>success</status>
    <message>
        <file-management>
            <list-files-result>
                <list-result>
                    <filename>test1.cfgtpl</filename>
                    <path>/opt/ConfigEngine/CSCOcnsie/Templates</path>
                </list-result>
                <list-result>
                    <filename>subdir/test2.cfgtpl</filename>
                    <path>/opt/ConfigEngine/CSCOcnsie/Templates</path>
                </list-result>
                <list-result>
                    <filename>subdir1/test3.cfgtpl</filename>
                    <path>/opt/ConfigEngine/CSCOcnsie/Templates</path>
                </list-result>
                <list-result>
                    <filename>test4.cfgtpl</filename>
                    <path>/opt/ConfigEngine/CSCOcnsie/Templates</path>
                </list-result>
            </list-files-result>
        </file-management>
    </message>
</cns-response>
```

Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE cns-response SYSTEM "d:\tfm\xml\cns_file_management.dtd"> -->
<cns-response>
    <message-id>1005</message-id>
    <status>failure</status>
    <message>
        <file-management>
            <files-result>
                <result>
                    <file-id>5</file-id>
                    <error-info>
                        <error-code>11</error-code>
                        <error-message>File Not Found</error-message>
                    </error-info>
                </result>
                <result>
                    <file-id>6</file-id>
                    <error-info>
                        <error-code>10</error-code>
                        <error-message>huh?</error-message>
                    </error-info>
                </result>
            </files-result>
        </file-management>
    </message>
</cns-response>
```

Example 3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE cns-response SYSTEM "http://boniface-ibm/config/cns_file_management.dtd">
<cns-response>
    <message-id>2</message-id>
    <status>success</status>
    <message>
        <file-management>
            <export-files-result>
                <export-result>
                    <result>
                        <file-id>1003</file-id>
                        <error-info>
                            <error-code>200</error-code>
                            <error-message>Successfully exported template file
'test4.cfgtpl'.</error-message>
                        </error-info>
                    </result>
                    <content>!
! test4.cfgtpl
!
</content>
                </export-result>
                <export-result>
                    <result>
                        <file-id>1004</file-id>
                        <error-info>
                            <error-code>200</error-code>
                            <error-message>Successfully exported template file
'DemoRouter.cfgtpl'.</error-message>
                        </error-info>
                    </result>
                    <content>!
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
service udp-small-servers
service tcp-small-servers
!
hostname DemoRouter
!
boot system flash c7200-is-mz
enable secret 5 $1$cMdI$.e37TH540MWB2GW5gMOn3/
enable password cisco
!
ip subnet-zero
!
interface FastEthernet0/0
 no ip address
 no ip directed-broadcast
 no ip route-cache
 no ip mroute-cache
 shutdown
 half-duplex
!
interface Ethernet1/0
 ip address 10.10.1.1 255.255.255.240
 no ip directed-broadcast
```

```
 no ip route-cache
 no ip mroute-cache
!
interface Ethernet1/1
 no ip address
 no ip directed-broadcast
 no ip route-cache
 no ip mroute-cache
 shutdown
!
interface Ethernet1/2
 no ip address
 no ip directed-broadcast
 no ip route-cache
 no ip mroute-cache
 shutdown
!
interface Ethernet1/3
 no ip address
 no ip directed-broadcast
 no ip route-cache
 no ip mroute-cache
 shutdown
!
ip classless
ip route 0.0.0.0 0.0.0.0 10.10.1.1
ip http server
!
dialer-list 1 protocol ip permit
dialer-list 1 protocol ipx permit
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password cisco
 login
!
end
</content>
               </export-result>
           </export-files-result>
       </file-management>
    </message>
</cns-response>
```

Limitations

The CE XML DTD has the following limitations:

- The <path> tag is NOT supported in Cisco Configuration Engine. The <path> tag is ignored and all template management operations are performed locally on the host file system in the directory $CISCO_CE_HOME/Te*mplates*.
- The <url> tag under source tag must be empty, which will signal that the content is in the payload <content>.
- According to the DTD, multiple files can be specified in each import/export/remove operation; however, for the Cisco Configuration Engine only a single file is handled in each operation.

Error Codes

The response XML includes error codes that indicate the status of the requested operation. The codes are listed in the table below.

Table 3-1 Template File Management Error Codes

Error Code	Error message	Description
200	Success	The request succeeded.
400	Request missing	No data was posted to the servlet.
401	Authentication required	Authentication missing or failed.
403	Deletion denied	The requested file could not be deleted because it does not exist or the user has insufficient permission.
404	Contains a list of missing required parameters	Missing required parameter.
405	Not supported or implemented	The request is either not supported or implemented.
406	File with specified filename already exists	The parameter passed is not acceptable because the filename specified already exists.
417	File not found	The requested file either does not exist or the user does not have access permission.
498	Invalid XML request	The XML file posted to the servlet is either badly formed or invalid.
499	Any	Any error other than ones described above
500	Internal server error	Unexpected internal error.

Examples

The following section contains Client, XML Request, and Response examples.

Client Examples

This section contains Client examples in HTML and Java.

HTML

Below is a simple HTML example of posting an XML file to the TFM (Template File Manager) servlet using the multipart/form-data MIME content type.

```
<HTML>
  <HEAD>
    <TITLE>
     Template File Manager Client
    </TITLE>
  </HEAD>
  <BODY>
    <CENTER>
      <FORM METHOD="POST"
            ENCTYPE="multipart/form-data"
            ACTION="http://beethoven/cns/CommandProcessor">
        <p>
          Browse to a <B>cns-request</B> XML document
          <br>to POST to the CommandProcessor:
        </p>
        <INPUT NAME="command" TYPE="file"><BR><BR>
        <INPUT TYPE="submit">
        <INPUT TYPE="reset">
      </FORM>
    </CENTER>
  </BODY>
</HTML>
```

JAVA

Below is a simple programmatic example of posting XML to the servlet using the multipart/form-data MIME content type. This simple java class PostUtil.java utilizes an HTTPClient class library from http://www.innovation.ch/java/HTTPClient The jar file HTTPClient.jar (which can be obtained from the above URL) should be in the classpath.

In this client example, you would be posting to your host system with hostname beethoven a <cns-request> XML document that you had saved in a file import.xml to the following URL:http://beethoven/cns/CommandProcessor

```
import HTTPClient.*;
import java.io.*;
import java.net.*;

public class PostUtil
{
    public static void main(String[] args)
    {
        String _sHost       = "beethoven";
        String _sPort       = "80";
        String _sPath       = "/cns/CommandProcessor";
        String _sParamName  = "command";
        String _sFileName   = "import.xml";

        try
        {
            String _sResponse = PostUtil.postFile(_sHost,
                                                  _sPort,
                                                  _sPath,
                                                  _sParamName,
                                                  _sFileName);
            System.out.println(_sResponse);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }


    /**
     *  This method uploads a file to a server via a HTTP post().
     *  The file content will be included as the value of a name/value pair
     *  encoded using the 'multipart/form-data' MIME content-type
     *  (and therefore the encoding) and submitted to a process on the server
     *  that will receive and parse the form data.
     *
     *  @param path - path to servlet/cgi that will process the post
     *  @param paramName - name in the name/value pair submitted in form
     *  @param uploadFileName - name of file whose content will be the value
     */
    public static String postFile(String host,
                                  String port,
                                  String path,
                                  String paramName,
                                  String uploadFileName)
    throws IOException, ModuleException
    {
        // arg validation code left out for brevity

        StringBuffer _sBuff = new StringBuffer();
        URL url = new URL("http://" + host + ":" + port);
```

```
        NVPair[] opts = { new NVPair("","") };
        NVPair[] file = { new NVPair(paramName, uploadFileName) };
        NVPair[] hdrs = new NVPair[1];
        byte[] data = Codecs.mpFormDataEncode(opts, file, hdrs);

        HTTPConnection con = new HTTPConnection( url );
        HTTPResponse rsp = con.Post( path, data, hdrs );

        if ( rsp.getStatusCode() >= 300 )
        {
            System.err.println( "Error: " + rsp.getReasonLine() );
            String errorMsg =  new String(rsp.getData());
            System.err.println( errorMsg );
            _sBuff.append(errorMsg);
        }
        else
        {
            BufferedReader _hIn = new BufferedReader(new InputStreamReader
                                                 (rsp.getInputStream()));
            String _sLine = null;

            while((_sLine = _hIn.readLine()) != null)
            {
                _sBuff.append(_sLine);
            }

            _hIn.close();
        }
        return _sBuff.toString();
    }
}
```

Note The method `Codecs.mpFormDataEncode(opts, file, hdrs)` encodes the form data for multipart/form-data content type.

XML Request Examples

Below are given three sample <cns-request> XML documents, one for each template operation.

Import Template "import.test.cfgtpl"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-request SYSTEM "http://beethoven/cns_file_management.dtd">
<cns-request>
    <message-id>2</message-id>
    <security>
        <username/>
        <password/>
    </security>
    <message>
        <file-management>
            <import-files>
                <import-file>
                    <file-id>1003</file-id>
                    <source-url>
                        <url/>
                    </source-url>
                    <target-file >
                        <filename>import.test.cfgtpl</filename>
                        <path/>
                    </target-file>
                    <overwrite>true</overwrite>
                    <content>!
! content of import.test.cfgtpl
!</content>
                </import-file>
            </import-files>
        </file-management>
    </message>
</cns-request>
```

Export Template "export.test.cfgtpl"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-request SYSTEM "http://beethoven/cns_file_management.dtd">
<cns-request>
    <message-id>2</message-id>
    <security>
        <username/>
        <password/>
    </security>
    <message>
        <file-management>
            <export-files>
                <export-file>
                    <file-id>123</file-id>
                    <source-file>
                        <filename>export.test.cfgtpl</filename>
                        <path/>
                    </source-file>
                    <target-url>
                        <url/>
                        <security>
                            <username/>
                            <password/>
                        </security>
                    </target-url>
                </export-file>
            </export-files>
        </file-management>
    </message>
</cns-request>
```

Remove Template "remove.test.cfgtpl"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-request SYSTEM "http://beethoven/cns_file_management.dtd">
<cns-request>
    <message-id>222</message-id>
    <security>
        <username/>
        <password/>
    </security>
    <message>
        <file-management>
            <remove-files>
                <remove-file>
                    <file-id>333</file-id>
                    <source-file>
                        <filename>remove.test.cfgtpl</filename>
                        <path/>
                    </source-file>
                </remove-file>
            </remove-files>
        </file-management>
    </message>
</cns-request>
```

XML Response Example

In the programmatic Java example above, if the XML file uploaded (import.xml) had the content shown in the Import Template "import.test.cfgtpl" example, a successful response would look like the following:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <cns-response>
  <message-id>2</message-id>
  <status>success</status>
   <message>
 <file-management>
<files-result>
<result>
  <file-id>1003</file-id>
<error-info>
  <error-code>200</error-code>
  <error-message>Successfully imported
template file
import.test.cfgtpl'.</error-message>
  </error-info>
  </result>
  </files-result>
  </file-management>
  </message>
  </cns-response>
```

Template Features

The configuration template can include simple control structures such as, *if*, *else* and *elseif*. By using these control structures, the user can include or exclude a block of CLI commands based on a parameter stored in the directory.

The syntax for these **#** preprocessing control structures is as follows:

Syntax Description

#if <*URL*> = *constant*

cli-command(s)

#elseif <*URL*> = *constant*

cli-command(s)

#else

cli-command(s)

#endif

Where *constant* is an integer, boolean or a string in single quotes and the <*URL*> is a URL pointing to an attribute in the Directory or Database.

Note Nested **#if**, **#elseif**, and **#include** is NOT supported.

The configuration template can include **#define** entries to define short names for long URLs.

The syntax for the **#define** preprocessing command is as follows

#define *definition-name <URL>* | *constant*

where *<URL>* is a reference to an attribute in the directory.

The configuration template can contain another **#** preprocessing command **#include,** which allows the inclusion of other configuration templates.

The syntax for the **#** preprocessing command is as follows:

#include *<URL>* | '*<Filename>*' | *<Filename>*

Whenever an **#include** directive is encountered, it is replaced by the content of the file.

The following configuration template sample includes either IP sub-template or ISDN sub-template based on the value of the parameter protocol in the directory or database.

Examples

```
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
service udp-small-servers
service tcp-small-servers
!
hostname ${LDAP://this:attrName=IOShostname}
#if ${LDAP://this:attrName=IOSIPprotocol} = true then
    #include ${LDAP://this:attrName=IPsubTemplate}
#else
    #include ${LDAP://this:attrName=ISDNsubTemplate}
#endif
```

The parameter, `${LDAP://this:attrName=IPsubTemplate}` contains the location of the file.

Dynamic Template and Object

Preparation

Log in to the Configuration Engine by means of SSH and issue the following command:

createDynamicLdapConf

It will ask for credential information for the external LDAP server. For example:

```
[root@agito-rm root]# createDynamicLdapConf
Enter Server Host Name or IP address: 10.1.17.201
Enter User DN: cn=dcdadmin,ou=ha-alpha1,o=csco,c=us
Enter User Password:
Configuration File: [/opt/ConfigEngine/CSCOcnsie/conf/10.1.17.201.dynamic.conf] is
created.
[root@agito-rm root]#
```

After this command is executed, a configuration file is generated under $CISCO_CE_HOME*/conf* with a suffix of: ***.dynamic.conf***. In the above example, **10.1.17.201.dynamic.conf** is generated. The Configuration Server application uses the information in this file to bind to an external directory.

For example, if the device connects to the server with the URL shown below, the Configuration Server looks for the **10.1.17.201.dynamic.conf** file in $CISCO_CE_HOME*/conf* and uses the information in the file to bind to the external LDAP server to access the specified object.

/cns/DynaConfig/cfgtpl=test.cfgtpl/object=ldap://10.1.17.201/cn=DemoRouter,ou=CNSDevices, ou=ha-alpha1,o=csco,c=us

URL to Connect to Configuration Engine

There are two URLs that the device can use to connect to the Configuration Engine:

Type 1

/cns/DynaConfig/cfgtpl=<*template name*>

For example:

/cns/DynaConfig/cfgtpl=test.cfgtpl

This URL contains a single parameter for the template filename. The Configuration Server uses the template referenced by the URL to generate the configuration and resolves any parameters in the template with attributes from the device object. If the value is filename only, Cisco Configuration Engine SDK looks at that filename under the default template directory: $CISCO_CE_HOME/*Template*.

Type 2

/cns/DynaConfig/cfgtpl=<*template name*>/object=ldap://<*server hostname or ip address*>/object=<*object location*>

For example:

/cns/DynaConfig/cfgtpl=test.cfgtpl/object=ldap://10.1.17.201/cn=DemoRouter,ou=CNSDevices,ou=ha-alpha1,o=csco,c=us

This URL contains two parameters; one is the template filename and the other is the URL of the object to use to resolve parameters in the template. The object URL contains the address for the LDAP server in which the object resides.

Example

Here is an example of the event payload to trigger the configuration download:

```
<?xml version="1.0" encoding="UTF-8"?>
<config-server config-action="write" no-syntax-check="TRUE">
    <identifier>DemoRouter-1053568055542</identifier>
        <server-info>
            <ip-address>alex-rm2.cisco.com</ip-address>
                <web-page>/cns/DynaConfig/cfgtpl=test.cfgtpl/object=ldap://10.1.17.201/
cn=DemoRouter,ou=CNSDevices,ou=ha-alpha1,o=csco,c=us</web-page>
        </server-info>
</config-server>
```

Error Handling

Error handling utilizes the existing error reporting mechanism with these additional error messages.

Error Condition: Template not found

Explanation: Error message is returned to the device containing the message:

Error Message: The desired template can not be found under /opt/ConfigEngine/CSCOcnsie/Template

Error Condition: Object not found

Explanation: Error message is returned to the device containing the message:

Error Message: The desired object can not be found in the data source

Error Condition: Parameter not found in attribute list of desired object

Explanation: Error message is returned to the device containing the message:

Error Message: Parameter(s) can not be found in the object: <*object name*>

Error Condition: LDAP Server Authentication Failure

Explanation: Given Ldap Server credential configuration is invalid.

Error Message: Authentication Failure

Restrictions

The credential information for the external LDAP servers need to be pre-configured in the Configuration Engine.

CHAPTER 4

Namespace Mapping Service

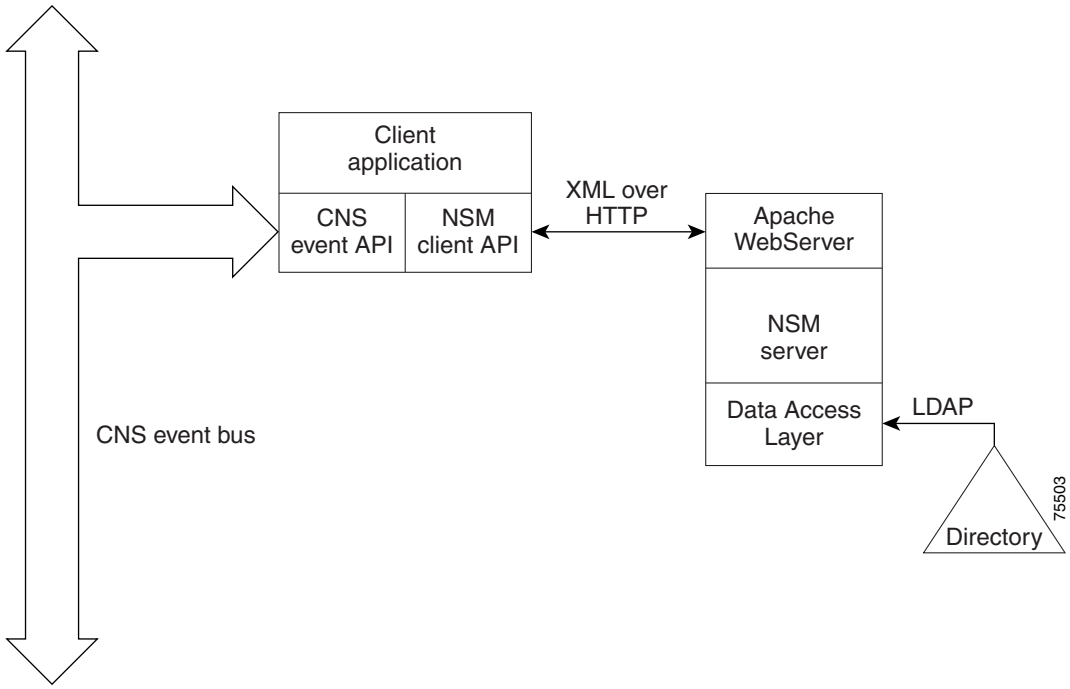
The Namespace Mapping Service (NSM) API allows you to address multiple network devices by a single posting of a publish or subscribe event, and it allows you to map Cisco-standardized event names to names of his or her choosing.

For example, in a network of 100 routers, there might be 10 which the administrator wants to configure as a VPN (Virtual Private Network). In order to load a configuration into each of these devices, your client application could either publish 10 *cisco.mgmt.cns.config.load.<deviceId>* events, or the administrator could associate the 10 devices with a common group name and your client application can post the event once. The administrator could rename the *cisco.mgmt.cns.config.load* subject to *application.load./config/westcoast* and group all the devices in the West Coast under a group called "westcoast." Then the application would just have to publish on *application.load./config/westcoast* and the devices in the "westcoast" group would get the event.

Namespace Mapper

The Namespace Mapper is a service that maps one subject to one or more subjects. The purpose of the Namespace Mapper is that it gives an application the ability to address a group of devices in a single event. It also de-couples the development namespace from the deployment (application) namespace.

Figure 4-1 Namespace Mapper architecture



The client application accesses the Namespace Mapper through a set of client APIs as seen in Figure 4-1. The client APIs contact the Namespace Mapper Server through XML messages sent over HTTP transport. The Namespace Mapper server runs in a web server environment and accesses the data store through a Data Access Layer. The data store supported in the current implementation is the Directory, a hierarchical data store for storing objects. The data store has information about groups, application namespaces, and subject mapping.

Mapping depends on the namespace that the application uses. The namespace consists of a set of subjects. One or more applications could use the same namespace. The mapping returned by the NSM depends on the namespace specified and the created application groups.

Namespace Mapper Operation

Let us consider the case of devices with ids R1, R2 and R3, which are connected to a Event Gateway as shown in Figure 4-2. (The Event Gateway acts as a relay between the device and the event bus, and acts on behalf of the device to publish or subscribe on the event bus). When a device makes a connection with the gateway, it sends its device identifier to the gateway, identifying itself. When the Event Gateway starts, it is passed an application parameter of **config**, and it attaches to the **config** namespace. The default namespace for Cisco Configuration Engine is **config**.

Devices R1 and R2 are grouped into a group called G1. G1 is assumed to be created within the namespace *config*. R3 does not belong to any group. The subject *config.load* has been mapped to *abcd.load* in the **config** namespace.

When the Gateway receives a request from the three devices to subscribe to config.load, the Gateway contacts the Namespace Mapper to resolve *config.load* in the **config** namespace.

The Namespace Mapper returns the following set of subscribing mappings to the Event Gateway, for the devices R1, R2 and R3.

Subscriber mapping for R1: abcd.load.R1, abcd.load./config/G1

Subscriber mapping for R2: abcd.load.R2, abcd.load./config/G1

Subscriber mapping for R3: abcd.load.R3

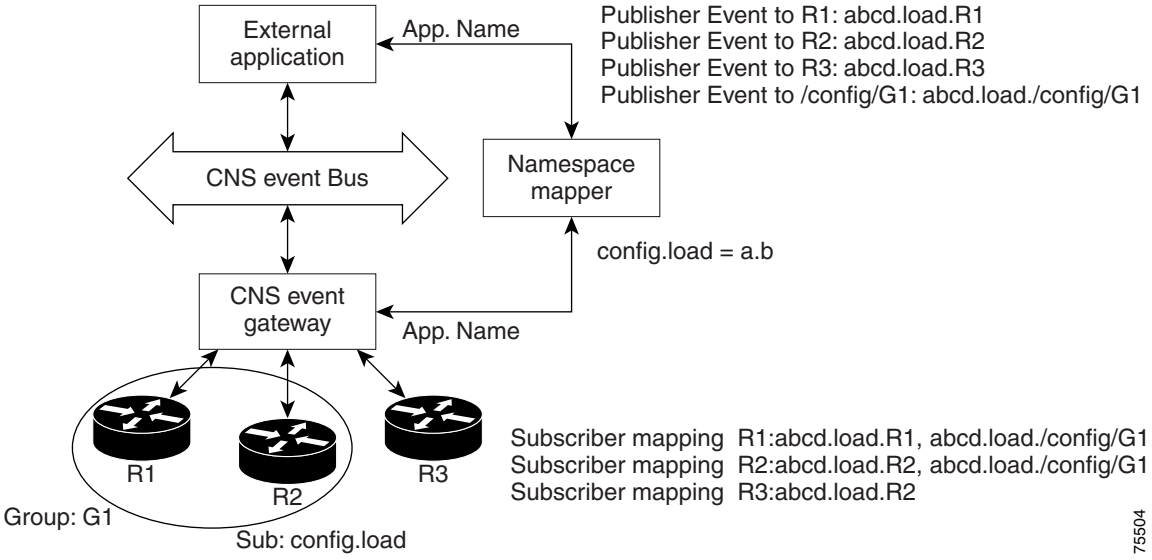
The mapping (*abcd.load*) for R1 and R2 are specialized with the deviceId and groupId, because R1 and R2 belong to a group, whereas the mapping for R3 has been specialized with just the deviceId. Note that */config/G1* is referred to as the absolute groupId, indicating that group G1 is created right under the **config** namespace.

When an external application wants to publish to either of the devices individually, or address them as a group, it calls the Namespace Mapper passing in an application name as parameter (**config**). In either case, the Namespace Mapper specializes the mapping (*abcd.load*) with the ID of the individual device or the absolute group Id.

For example, if an external application wants to send a *config.load* to device R2, it would query Namespace Mapper for the resulting map *abcd.load.R2* to publish.

Alternatively, if an external application wants to send a *config.load* to the group */config/G1* (i.e., to both devices R1 and R2), it would query Namespace Mapper for the resulting map *abcd.load./config/G1* to publish.

Figure 4-2 Example of NSM operation



Namespace Mapper (NSM) Client API

The NSM Client API provides an interface to its clients for querying the subscriber and publisher maps.

The methods of the NSM API are the same in both Java and C++. They are contained in a single class, **NSMClient**, as follows:

```
NSMClient — C++ only
attach
detach
resolve
```

NSMClient

NSMClient Constructor, used only if C++ is used, creates an NSMClient object. It takes one argument, which configuration file to use for this instance of NSMClient.

The NSM client looks for the configuration file in the following order for C++:

- NSM looks for the file specified in the constructor in the path specified in the NSM_PATH environment variable; for example, $NSM_PATH/./nsm.conf (see Appendix B for a complete listing of this file).
- If the file is not found, NSM looks for it in the current directory.

For Java API, the client looks for *nsm.conf* file in the following order:

1. In the path or locations specified in the CLASSPATH environment variable. The path or locations are searched in the order that they are specified.
2. In the current running directory

Note You can specify the location of *nsm.conf* file either through the environment variable NSM_PATH, or NSMClient call, but not both.

attach

The **attach()** call ties the client application to the specified namespace. All subsequent calls will be handled in the context of that namespace. The **attach()** operation takes a single argument—the name of the network application (application namespace).

detach

The **detach()** method "detaches" the application from the namespace. Call the **detach()** method only when you are finished resolving namespace mappings.

resolve

The **resolve()** operation provides the mechanism by which it actually resolves the original subject into one or more subjects, based on the namespace to which the client is "attached". The operation takes five arguments:

- **Output Parameter** — a mapped subject list into which the NSM servlet can write the results of the **resolve()** call
- **Entity ID** — unique name of the device or group of devices for which mapping is requested
- **Subject** — the subject for which mapping is requested
- **Action** — the action to be taken (subscribe or publish)
- **Element** — indicates if mapping is requested for a device or a group

Namespace Mapper Client Modes

The subject namespace has been modified in accordance with the new Cisco subject naming conventions. To keep up with the new naming convention, agents in Cisco IOS have been modified and released with the Cisco IOS 12.3 train. When the application "attaches" to the Namespace Mapper Client object, the client looks up a configuration file called *nsm.conf*.

The Namespace Mapping service operates in what is called: Provider Mode:

- Algorithmic — NSM server uses a mapping algorithm

- Non-algorithmic — NSM server mapping algorithm is overridden by the application

In the provider mode a **resolve()** request is handled by the NSM server. The Namespace Mapping service contacts the server, and the server returns the mapping list, based on the objects and their associations set up in the data store. In this mode, the complete URL of the Namespace Mapper Server is given in *nsm.conf*, for example:

http://<*hostname*>:80/nsm/NSMServlet

The following steps are required for the client application to communicate with agent-enabled Cisco IOS devices:

- The Event Gateway should be running, and be given the right application namespace as parameter. The application namespace should be exactly the same as the application namespace object in the directory. The reason is because, the gateway, as an application of the Namespace Mapper would "attach" to that namespace. It is important for the subscribing application (event gateway) and the publishing application (customer application) to "attach" to the same namespace for messaging to work. By default, on the host systems, the Event Gateway will be started with the application namespace **config**. So, the server would expect an application namespace object named **config** in the directory. The customer application should also attach to the **config** namespace. If for some reason, you wish to change **config** to some other name, the Gateway should be started with the same parameter, which can be specified through host setup.
- The client side configuration file *nsm.conf* should contain the complete URL of the Namespace Mapper Server.
- If the file is not present, the Namespace Mapper assumes default or append mode.
- NSM objects should be created in the host data store (Internal/External). The directory should be populated in accordance with the object model described in the Programmer guide. The host system should be configured to operate NSM in provider mode.

Note The Event Gateway and the Namespace Mapper Server currently reside in the host system.

Namespace Mapper Server

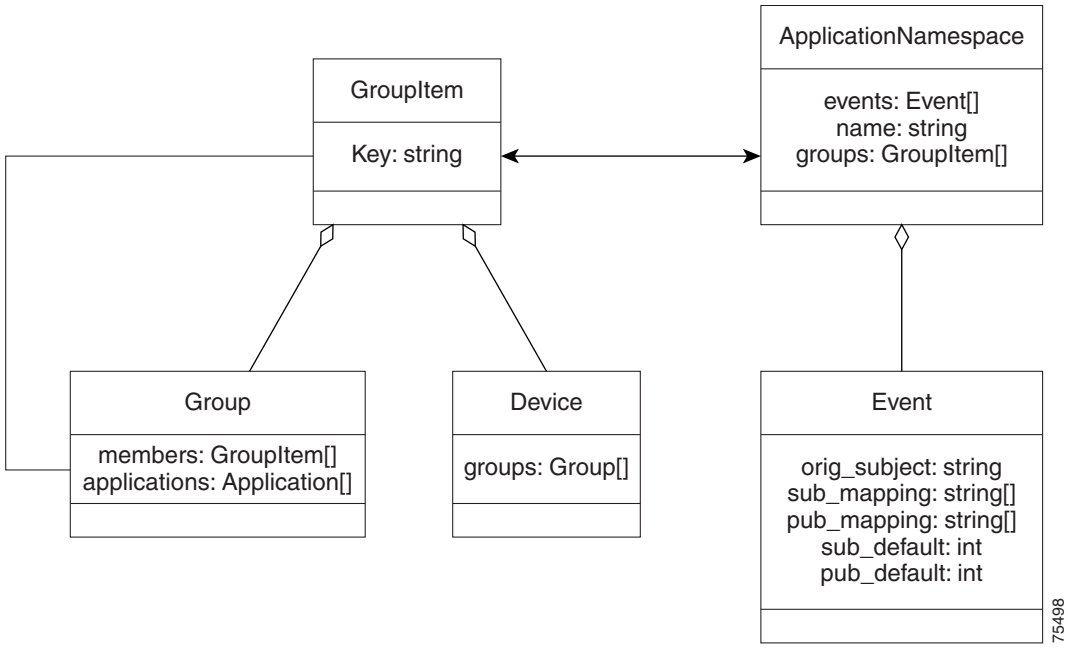
When the client operates in the Provider mode, the Namespace Mapper server is contacted for resolving the original subject. The server then looks up the entries in the directory to determine the mapping. The object model and the logic that the Namespace Mapper server follows is discussed in the next section.

Object Model

The following object model has been translated into directory schema and the directory should have the structure shown in the object model in Figure 4-3. It is the administrator's responsibility to populate the directory accordingly.

The model contains device, group, application and event classes. Devices in the network are grouped into logical groups based on criteria such as geographic location, device type etc. A group is an aggregation of one or more devices or other groups. Groups are associated with one or more application namespaces. An application namespace consists of one or more events. The Event objects determine how the subject should be mapped.

Figure 4-3 NSM Object Model



GroupItem

GroupItem is the base class for Group and Device, with an attribute called **key**, which is inherited by the Group and Device class.

Device

A device object represents a device in the network. Every device has a unique identifier. The **key** attribute of the GroupItem class represents this identifier. A device object should also have an attribute called groups, which points to all the groups containing the device.

Group

A group object represents a group of devices or other groups. The **members** attribute refers to the members of the group. The **key** attribute of the group object uniquely identifies the group. The **applications** attribute specifies the application namespaces with which the group is associated.

For example, all the devices located in California could be grouped into a group identified as **california**. A VPN application using the **config** application namespace could use the **california** group. Thus, the **california** group must have an association with the application namespace called **config**.

Application Namespace

An application object represents an application namespace. An application namespace is composed of one or more events specified in the **events** attribute. An application is associated with one or more groups, and the groups attribute gives the list of group associations.

Continuing the previous example, the **config** application namespace would have events like *cisco.mgmt.cns.config.load*, *cisco.mgmt.cns.config.complete*, *cisco.mgmt.cns.complete.failure*, etc. The **config** application namespace would also have associations with one or more groups of devices.

Consider the following example. A device with identifier D1 belongs to the **california** group and another device D2 belongs to the **newyork** group. An application using the **config** namespace wants to publish to all the devices belonging to both the groups (**california** and **newyork**). For the event to reach the devices, an association must exist between the **config** namespace and the two groups.

Event

The **orig_subject** attribute refers to the subject to be mapped. The **sub_mapping** and **pub_mapping** attributes give the list of event mapping for a subscriber and producer, respectively. The **sub_default** and **pub_default** attributes specify whether the namespace mapper server should use its algorithm to resolve, or whether it should be overridden.

If the **sub_default** and **pub_default** attributes are set to 1, it indicates that the server algorithm should be used. If this is the case, the server specializes the mapping listed in **sub_mapping** and **pub_mapping** with the identifier of the device or group for which mapping is requested.

If the **sub_default** and **pub_default** are set to 0, it indicates that the **sub_mapping** and **pub_mapping** attributes returns the absolute mapping list and no further specialization is needed.

Let us consider the example of the *ciso.cns.config.load* event. Let us say that it is mapped to *abcd.config.load* for both a publisher and a subscriber, and the user wants the server to use its algorithm for mapping. In this case, the **orig_subject** attribute is *cisco.mgmt.cns.config.load*, the **sub_mapping** and **pub_mapping** attributes are *abcd.config.load*. The **sub_default** and **pub_default** attributes are set to **1**.

Schema Description

This section describes the schema (see Appendix B for a listing of the example schema) extensions for the Namespace Mapper. It also specifies the actual directory schema objects that correspond to the classes in the Object model for the Namespace Mapper.

Device

In terms of the device, Namespace Mapper does not define the schema; however, Namespace Mapper does require two attributes to be in the customer's device object schema.

One of the attributes uniquely identifies the device. This attribute corresponds to the key field of the GroupItem class in the object data model. In this illustration, we will call this attribute **id**.

The other attribute refers to the group objects that the device belongs to. This corresponds to the groups attribute of the Device class in the object model. In this illustration, we will call this attribute **groups**.

The actual attribute names that the user defines are read in from a properties file, that can be edited through the host setup program. The Device Id and Group Attribute inputs of the host setup program correspond to **id** and **groups** respectively.

Group

In past releases, the class **cnsesGroup** was defined as the Group class of the object model. It is derived from the **groupofnames** class, and does not support hierarchical grouping.

Starting from Cisco Configuration Engine, the new class **cnsGroup** is defined as the Group class of the object model. It is derived from the **OrganizationalUnit** class. It inherits the nesting capability of **OrganizationalUnit** and allows hierarchical groups to be defined. The class **cnsGroup** contains a **cnsMember** attribute, which is a multi-value string attribute that holds the DNs of the associated device members.

Application Namespace

The object data model also mentions the association between **GroupItem** and the application namespace object.

The application namespace class is an **OrganizationalUnit**. An association relation between the namespace and group was implemented in the past, using two reference attributes.

Starting from Cisco Configuration Engine, a containment relation is used. Since Namespace is defined using **OrganizationalUnit**, it can contain other type of **OrganizationalUnit** such as group. All of the groups that are associated with a namespace are now created inside the namespace container.

Event Object

The *cnsesEvent* class represents the Event class in the object model. To achieve the behavior described by the object data model, following attributes were added to the schema:

- **cn** — original subject
- **cnsessubscribermap** — contains the subject(s) the original subject should be mapped to for a subscriber.
- **cnsespublishermap** — contains the subject(s) the original subject should be mapped to for a publisher.
- **cnsessubscriberdef** — for a subscriber, determines whether Namespace Mapper should further specialize the contents of cnsessubscribermap.
  - Algorithmic mode — Valid values are 1 and 0 with 1 signifying algorithmic mode.
- **cnsespublisherdef** — for a publisher, determines whether Namespace Mapper should further specialize the contents of cnsespublishermap.
  - Algorithmic mode — Valid values are 1 and 0 with 1 signifying algorithmic mode.

The Event objects are contained within the namespace organizational units, since events are specific to Namespaces.

Table 4-1 Schema extension to the Namespace Mapper

Class	Attributes	Attribute Type	Multi-valued	Description
Device* represents Device Class	Id*	Directory String	No	Uniquely identifies device object.
	Groups*	dn	Yes	Refers to the group objects to which that device belongs.
OrganizationalUnit represents Application Namespace	ou	Directory String	No	Name of the application namespace.

Table 4-1 Schema extension to the Namespace Mapper (continued)

Class	Attributes	Attribute Type	Multi-valued	Description
cnsesEvent represents an Event	cn	Directory String	No	Original subject to be mapped.
	cnsesSubscriberMap	Directory String	Yes	Mapped Subjects for a subscriber.
	cnsesPublisherMap	Directory String	Yes	Mapped Subjects for a publisher.
	cnsesSubscriberDef	int	No	Specifies algorithmic mode (value=1) or non-algorithmic mode (value=0) for subscriber.
	cnsesPublisherDef	int	No	Specifies algorithmic mode (value=1) or non-algorithmic mode (value=0) for publisher.
cnsGroup	cnsMember	dn	Yes	Refers to the device objects belonging to this group. This class is derived from objectClass OrganizationalUnit (ou).

Mapping Algorithm

The Namespace Mapper server looks up the directory for the objects and associations between them. It implements a different algorithm for a subscriber and a publisher.

Subscriber Logic

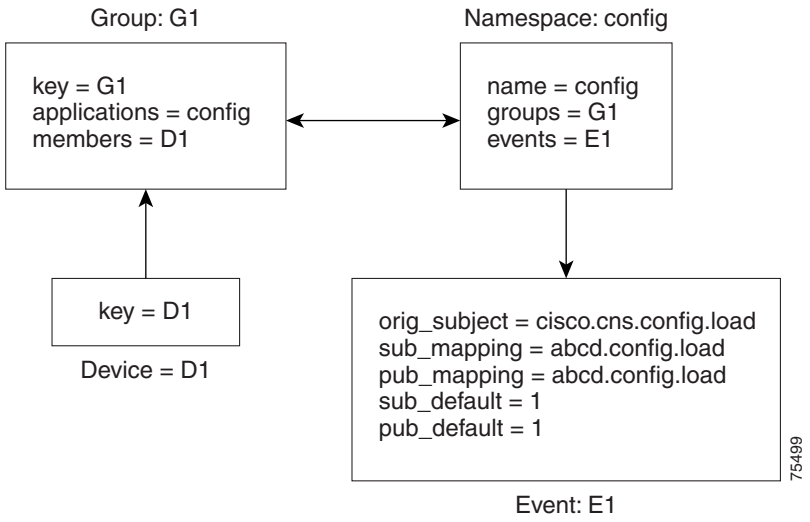
Let us consider the case of a device D1 (see Figure 4-4), which is connected to a Event Gateway. (The Event Gateway acts as a relay between the device and the event bus, and acts on behalf of the device to publish or subscribe on the event bus). When the device connection is established between the device and gateway, it sends its device identifier to the Event Gateway, identifying itself. When the Event Gateway is started upon start of the host, it is passed an application parameter of **config** by default and attaches itself to the **config** namespace.

Let us assume that the device D1 belongs to a group /config/G1 where the group G1 is created in the **config** namespace. Within the **config** namespace, *cisco.mgmt.cns.config.load* is mapped to *abcd.config.load*.

When the Event Gateway receives a subscription request from the device D1 to subscribe for *cisco.mgmt.cns.config.load*, the gateway, which is attached to the namespace mapper with the **config** namespace, calls **resolve**, passing in the necessary parameters. The device id would be D1, the action would be subscribe, and the subject to be mapped would be *cisco.mgmt.cns.config.load*.

The namespace mapper client, in the Provider mode, would then contact the Namespace mapper server, passing in the request to resolve, along with the required parameters.

Figure 4-4 NSM mapping algorithm



The server follows the following steps to determine the mapping.

Step 1 It finds the group(s) to which the device belongs in the namespace **config**. In this example, D1 belongs to the group /config/G1.

Step 2 Looks up the mapping for the action (publish/subscribe) in the namespace specified. In the **config** namespace, *cisco.mgmt.cns.config.load* is mapped to *abcd.config.load* for a subscriber.

Step 3 Looks up whether algorithmic mapping is specified. In this example, *sub_default* is 1, therefore, algorithmic mapping is desired.

Step 4 Specializes the value of *sub_mapping* with the device ID and group ID. The subject *abcd.config.load* is specialized to *abcd.config.load.D1* and *abcd.config.load./config/G1*.

Step 5 Returns the mapping to the client.

Step 6 If non-algorithmic mode is specified, *sub_default* is set to 0. The mapping list then becomes *abcd.config.load*.

Publisher Logic

Consider the case where an application A1 wants to publish an event on *cisco.mgmt.cns.config.load* to all the devices in the group /config/G1. The application would "attach" itself to the **config** namespace, and call **resolve** on the namespace mapper client. The group ID would be /config/G1, the action would be **publish**, and the subject would be *cisco.mgmt.cns.config.load*. The namespace mapper client, in the Provider mode, would then contact the Namespace mapper server, passing in the request to resolve, along with the required parameters. The server would take the following steps to determine the mapping:

Step 1 Check whether the group /config/G1 exists in the **config** namespace.

Step 2 Look up the mapping for the action (publish/subscribe) in the namespace specified. In the **config** namespace, *cisco.mgmt.cns.config.load* is mapped to *abcd.config.load* for a publisher.

Step 3 Look up whether algorithmic mapping is desired. In this example, **pub_default** is 1. Thus, algorithmic mapping is desired.

Step 4 Specialize the value of **pub_mapping** with the identifier. The subject *abcd.config.load* would be specialized to *abcd.config.load./config/G1*.

Step 5 Return the mapping to the client.

Step 6 If non-algorithmic mode were desired, **pub_default** would be 0. The mapping list would have been *abcd.config.load*.

Default Namespace *config*

Cisco Configuration Engine comes with a system namespace known as "config," which contains the set of Cisco standardized events, such as *com.cisco.cns.mgmt.config.load*. By default, each event defines a mapping to itself for both the publish and subscribe mapping.

The system namespace is guaranteed to return a mapping even for undefined events, in which case the input map is returned as the output map. This is a requirement for supporting future devices that might depend on new events that are not currently defined.

Administration and Setup

The first step is to set up the directory with Device, Group, Application Namespace and Event objects.

The next step is to set up configuration files for the Namespace Mapper. There are configuration files on both, the client and server side.

Client side configuration involves setting up the *nsm.conf* file in the directory where the client application runs. A template file (**nsm.conf**) is provided to simplify client side configuration.The directives for the *nsm.conf* file have been discussed in the section on client modes.

The C++ client library tries to locate *nsm.conf* passed in the NSMClient constructor. If it cannot find it, then it is prepend with the path specified in the environment variable NSM_PATH. The Java client library tries to locate *nsm.conf* in the path specified in the CLASSPATH. If it cannot find it, it looks for *nsm.conf* in the current directory.

Server configuration includes setting up property files for the Namespace Mapper server and the Data Access Server. This should be done using the host setup program.

The Event Gateway on host is started off with an application parameter. This is prompted during host setup as "Enter Event Gateway Application Parameter." The default value for this parameter is "config." For every application parameter entered here, there must be a corresponding Application Namespace object in the directory. Under the application namespace there must be event objects that correspond to subjects in that namespace. For example:

Under the "config" application the event objects are as listed below.

cisco.mgmt.cns.config.load

cisco.mgmt.cns.config.failure

cisco.mgmt.cns.config.complete

cisco.mgmt.cns.config.warning

NSM API Usage

Cisco network devices must be configured to publish or subscribe to individual events using Cisco IOS syntax (such as *cisco.mgmt.cns.config.load*).

When the device R1 signals to the Event Gateway that it wants to subscribe to an event, the gateway will use the NSM API to retrieve all appropriate namespace mappings—the customized event strings created by the network administrator—from the directory (such as *westcoast.load* or *eastcoast.load,* as appropriate for that device). The Event Gateway will then subscribe to the mapped subjects on the Event Bus.

When you want to post an event from your client application, you use the NSM API to retrieve all appropriate namespace mappings for the event, and then you post the mapped events to the Event Bus.

For example, when you want publish a *cisco.mgmt.cns.config.load* event to all of the devices in the */config/california* group, which are associated with the *config* application namespace, you would first "attach" to the *config* namespace, then invoke the **resolve()** operation with the string *cisco.mgmt.cns.config.load* as the *event* argument and the string "*/config/california*" as the *dev_id* argument.

The list of custom events that are mapped to the *cisco.mgmt.cns.config.load* and which are appropriate for devices in the */config/california* group will be returned in the *result* argument.

You would then publish the events returned in the *result* argument to the Event Bus using the Event API.

See Example 4-1 (C++) and Example 4-2 (Java). More examples with the Event and Namespace Mapper APIs are listed in the appendix.

Example 4-1 Mapping event strings in C++

```
#include <stdio.h>
#include <iostream.h>
#include "nsm_client.h"

void main()
{
    NSMClient *nsmc = new NSMClient ("./nsm.conf");
    NSMResult *result = new NSMResult();
    if (nsmc != NULL && result != NULL) {
        nsmc->attach("config");
        nsmc->resolve(*result, "/config/california", "cisco.cns.config.load",
    NSMClient::PUBLISH, NSMClient::GROUP);
    nsmc->detach();

    // Retrieve the event strings and, post to the event bus---or, as in this example,
    // output to screen.

    for (NSMResultIterator iter(*result); !iter.isEnd(); iter.advance())
    {
        cout << iter.value() << endl;
        }
    delete result;
    delete nsmc;
    }
}
```

Example 4-2 Mapping event strings in Java

```
import com.cisco.cns.nsm.client.*;

public class PublishSample
{
    public static void main(String args[])
    {
        NSMClient nsmc = new NSMClient("./nsm.conf");
        NSMResult result = new NSMResult();

        if (nsmc != null && result != null) {
            nsmc.attach("config");
            nsmc.resolve(result, "/config/california", "cisco.cns.config.load",
        NSCMLient.PUBLISH, NSMClient.GROUP);
        nsmc.detach();
        // Retrieve the event strings and post to the event bus, or, as in this example,
        output to screen.
        NSMResultIterator iter = new NSMResultIterator(result);
        for (; !iter.isEnd(); iter.advance())
            System.out.prntln((String)iter.value());
        }
    }
}
```

The NSMServer will retrieve the appropriate namespace mappings from the directory. These event strings will be contained in the *result* NSMResult object. The NSMResult object can hold an unlimited number of individual event strings. To retrieve all of the event strings in NSMResult, use the NSMResultIterator to cycle through the events.

If your network administrator had configured device R1 to belong to group */config/california*, and mapped the *cisco.mgmt.cns.config.load* event to the string *westcoast.load*, the returned NSMResult object would contain the following for a publisher:

```
westcoast.load./config/california
```

For device R1, which is a subscriber, NSM would have returned the following mapping in the NSMResult object:

```
westcoast.load.R1
westcoast.load./config/california
```

Your client application would then use the Event API to publish the events using the subjects contained in the NSMResult object of the Event Bus. The Event Gateway would receive these events, use the NSM API to map the events back to the *cisco.mgmt.cns.config.load* event, and pass it on to the R1, R2 and R3 devices, which had previously notified the gateway that they wanted to subscribe to the *cisco.mgmt.cns.config.load* event.

For complete, working examples in both C++ and Java of how to retrieve mappings from the data store using arguments input on the command line, see Appendix A, "Code Samples"

NSM SDK Contents

The NSM API is provided as both a C++ library and a Java *.jar* file.

Constants

The API defines several constants which you can use in your client application:

Java

- Arguments to the **resolve()** method:

```
public static final int DEVICE
public static final int GROUP

public static final int SUBSCRIBE
public static final int PUBLISH
```

- Return values:

```
public static final int SUCCESS
public static final int FAILURE
```

C++

- Arguments to the **resolve()** function:

```
enum (DEVICE, GROUP);
enum (SUBSCRIBE, PUBLISH);
```

- Return values:

```
enum (SUCCESS, FAILURE);
```

Supported CPP Compilers

Solaris

SUN CC6.0

Solaris Version 2.10

Solaris C++

Information on file location, linking instructions and compiler input, are found in Appendix A, "Code Samples." using the Event API and the Namespace Mapper API. Your LD_LIBRARY_PATH environment variable should reference the directories identified in Table 4-2.

Table 4-2 LD_LIBRARY_PATH

Directory	Contents
`/<INSTALLDIR>/CSCOesdk/lib`	If using CC6.0

Java

To use the Java class, set your classpath to reference the files in Table 4-3.

Table 4-3 CLASSPATH

File	Contents
`/<INSTALLDIR>/CSCOesdk/java/HTTPClient.jar`	Cisco Configuration Engine HTTP client
`/<INSTALLDIR>/CSCOesdk/java/NSMClient.jar`	Cisco Configuration Engine Namespace Mapping Service client
`/<INSTALLDIR>/CSCOesdk/java/xerces.jar`	SAX-driven XML parser

Import the package *com.cisco.cns.nsm.client* into your *.java* source file. The package is contained in the file *NSMClient.jar*.

NSM Client API Reference

This section documents the interfaces to the Namespace Mapping (NSM) Service. These interfaces provide a mechanism by which you can address groups of devices with a single event; you can also map Cisco standard event strings to strings that are more meaningful within your own network environment.

class NSMClient

The NSMClient class encapsulates the many of the functions of the NSM API.

Public Functions

NSMClient()

Constructor for NSMClient.

Usage:

```
NSMClient(const char *location = NSM_CONF_FILE);
```

Description:

Constructor for NSMClient. This is the preferred way of creating the NSMClient object.

Parameters

Location – location of the *nsm.conf* file. NSM_CONF_FILE is defined as ./nsm_conf. If this file is not found in the NSM_PATH directory then NSM looks in the current directory.

create()

Use this function to create an NSM Client.

Usage:

```
static NSMClient* create(void);
```

Description:

A static method which creates a NSMClient object and returns it back to the caller. This method is deprecated.

Returns

Pointer to an NSMClient object.

attach()

Attach to a namespace.

Usage:

```
virtual int attach(const char *application = NSMClient::DEFAULT);
```

Description:

All subsequent calls will be handled in the context of that namespace. The attach() operation takes a single argument—the name of the network application (application namespace).

Parameters

Application – the name of your network application namespace.

Returns

SUCCESS or FAILURE

detach()

Detach from namespace.

Usage:

```
virtual void detach(void);
```

Description:

The method "detaches" the application from the namespace. Call the detach() method only when you are finished resolving namespace mappings.

resolve()

The method provides the mechanism by which it actually resolves the original subject into one or more subjects, based on the namespace to which the client is "attached."

Usage:

```
virtual int resolve(char **&result,
const char *dev_id,
const char *event,
int action,
int element);
```

Description:

After the resolve() function has returned, the NULL-terminated result array will contain all of the event strings that have been mapped by your network administrator to the string in the event argument and that are applicable to the device identified by the dev_id argument.

The function will return FAILURE if any one of the following is true:

- Action is not PUBLISH or SUBSCRIBE
- Element is not DEVICE or GROUP
- Result is NULL

Parameters

Result – a NULL-terminated char array (created by the Provider object) containing the mapped events after the function returns. This memory must be deleted by the calling application.

dev_id – string identifying the device or group (key string of the GroupItem object in the Namespace object model).

event – the Cisco IOS event for which you want to obtain mapped event strings

action – SUBSCRIBE or PUBLISH.

element – indicates whether dev_id is a DEVICE or a GROUP.

Returns

SUCCESS or FAILURE

resolve()

The method provides the mechanism by which it actually resolves the original subject into one or more subjects, based on the namespace to which the client is "attached." The result is returned in the NSMResult object.

Usage:

```
virtual int resolve(NSMResult &result,
const char *dev_id,
```

```
const char *event,
int action,
int element);
```

Description:

After the resolve() function has returned, the NSMResult object will contain all of the event strings that have been mapped by your network administrator to the string in the event argument and that are applicable to the device identified by the dev_id argument. The function will return FAILURE if any one of the following is true:

- Action is not PUBLISH or SUBSCRIBE
- Element is not DEVICE or GROUP
- Result is NULL

Parameters

result – a NSMResult object containing the mapped events after the function returns. Application needs to construct this object and pass in to resolve().

dev_id – string identifying the device or group (key string of the GroupItem object in the Namespace object model).

event – the Cisco IOS event for which you want to obtain mapped event strings.

action – SUBSCRIBE or PUBLISH

element – indicates whether dev_id is a DEVICE or a GROUP.

Returns

SUCCESS or FAILURE

resolve()

The method provides the mechanism by which it actually resolves the original subject into one or more subjects, based on the namespace to which the client is “attached.” This uses the NSMResult and NSMResolveRequest objects.

Usage:

```
virtual int resolve(NSMResult &result,
const NSMResolveRequest &request);
```

Description:

After the resolve() function has returned, the NSMResult object will contain all of the event strings that have been mapped by your network administrator. This is a variant of the above resolve() call. The request parameters are pass in to NSMResolveRequest object. The function will return FAILURE if any one of the following is true:

- NSMResolveRequest.action is not PUBLISH or SUBSCRIBE
- NSMResolveRequest.element is not DEVICE or GROUP

Parameters

result – a NSMResult object containing the mapped events after the function returns. Application needs to construct this object and pass in to resolve().

request – a NSMResolveRequest object that contains the dev_id, event, action and element.

Returns

SUCCESS or FAILURE

class NSMResolveRequest

The NSMResolveRequest class stores the resolve request attributes.

Public Functions

NSMResolveRequest()

Constructor for NSMResolveRequest.

Usage:

```
NSMResolveRequest();
```

Description:

Constructor for NSMResolveRequest.

NSMResolveRequest()

Constructor for NSMResolveRequest.

Usage:

```
NSMResolveRequest(const char *dev_id, const char *event, int action, int element)
```

Description:

Constructor for NSMResolveRequest.

Parameters

dev_id – string identifying the device or group (key string of the GroupItem object in the Namespace object model).

event – the Cisco IOS event for which you want to obtain mapped event strings.

action – SUBSCRIBE or PUBLISH.

element – indicates whether dev_id is a DEVICE or a GROUP.

get_dev_id()

Get device id

Usage:

```
const char *get_dev_id(void) const;
```

Description:

Get device id.

Returns

Pointer to a dev_id.

get_event()

Get event string.

Usage:

```
const char *get_event(void) const;
```

Description:

Get event string. Event string is the subject name.

Returns

Pointer to an event string.

get_action()

Get action string.

Usage:

```
int get_action(void) const;
```

Description:

Get action string.

Returns

Action which can be NSMClient::SUBSCRIBE or NSMClient::PUBLISH.

get_element()

Get element.

Usage:

```
int get_elment(void) const;
```

Description:

Get element.

Returns

Element which can be NSMClient::DEVICE or NSMClient::GROUP.

set_dev_id()

Set device id

Usage:

```
void set_dev_id(const char *dev_id);
```

Description:

Set device id.

Parameters

dev_id – pointer to an dev_id

set_event()

Set event string.

Usage:

```
void set_event(const char *event)
```

Description:

Set event string. Event string is the subject name.

Parameters

event – pointer to an event string

set_action()

Set action string.

Usage:

```
void set_action(int action);
```

Description:

Set action string.

Parameters

Action – which can be NSMClient::SUBSCRIBE or NSMClient::PUBLISH.

set_element()

Set element.

Usage:

```
void set_element(int element);
```

Description:

Set element.

Parameters

Element – which can be NSMClient::DEVICE or NSMClient::GROUP.

class NSMResult

The NSMResult class contains all of the event strings that have been mapped. This object is filled in the NSMClient::resolve() function.

Public Functions

NSMResult()

Constructor for NSMResult.

Usage:

```
NSMResult();
```

Description:

Application needs to construct this object and delete it when it is no longer used. Application needs to construct this before and pass in to NSMClient::resolve().

get_num_items()

Return the number of mapped strings

Usage:

```
int get_num_items(void) const;
```

Description:

Return the number of mapped strings.

Returns

The number of mapped strings.

class NSMResultIterator

The NSMResultIterator class provides iterating events in NSMResult.

Public Functions

NSMResultIterator()

Constructor for NSMResultIterator.

Usage:

```
NSMResultIterator(NSMResult &result);
```

Description:

Application use this for the iterating mapped events in NSMResult.

Parameters

result – NSMResult object filled in by the NSMClient::resolve() function.

advance()

Advance to the next mapped string

Usage:

```
void advance();
```

Description:

Advance to the next mapped string.

value()

Return the current string.

Usage:

```
char* value(void);
```

Description:

Return the current string.

Returns

The current string. NULL if there are no more.

isEnd()

Return whether the iteration has completed.

Usage:

```
bool isEnd(void);
```

Description:

Return whether the iteration has completed.

Returns

True if it is done. false if there are more.

CHAPTER 5

Web Services: Admin, Config, Image, Exec, NSM

The Cisco Configuration Engine provides the ability to manage devices; specifically, the configurations and images on large numbers of IOS and non-IOS devices. External applications wishing to interact with the Cisco Configuration Engine might do so by means of a number of APIs defined in this document. This chapter explains how to use the Web Services interfaces to the Cisco Configuration Engine.

The following Web Service interfaces are provided:

- **ConfigService** – send/acquire configurations to/from devices.
- **ImageService** – delete files, obtain an inventory of the hardware, file system(s) & their content, distribute or activate image(s) on devices.
- **ExecService** – supports interactive commands including **show** commands and **reboot** on devices.
- **AdminService** – create and manage the various system objects used by the Cisco Configuration Engine to manage devices (such as devices, line-cards, images, configurations (templates), users, conditions, groups, passwords).
- **NSMService** – create and manage namespace, subjects in namespace and subject mappings in Namespace. It also includes an operational API to resolve subjects.

Web Services Model

Web Services utilize a publish, find, and bind model. For example:

1. Publish: Service Provider can publish a description of a Web Service to a Service Registry (for instance, a UDDI Repository).
2. Find: Service Consumer can search and find a Service and obtain its location.
3. Bind: Service Consumer then can bind to an instance of the Web Service.

Cisco Configuration Engine does not provide the Publish or Find parts of the model, only the endpoints that clients can Bind to as it is assumed clients know the service endpoints (service URLs) *a priori*.

The service endpoints are:

http://hostname/cns/services/CEConfigService

http://hostname/cns/services/CEImageService

http://hostname/cns/services/CEExecService

http://hostname/cns/services/CEAdminService

http://hostname/cns/services/CENSMService

Appending **?wsdl** provides the URL of the service's interface defined in Web Service Definition Language (WSDL), if Cisco Configuration Engine has been configured to expose the **wsdl**. The transport for this release of Web Services is HTTP (v.1.1).

Configuration Engine Web Services Overview

The ConfigService and ImageService provide a job semantic, whereby work is performed within the context of a job, each uniquely identified by an identifier string called the "job ID." This job ID is then used as a token to query the status of the job or manage the job's life cycle. Notification by email whenever a job completes, fails or is cancelled is available.

The first three service interfaces are operational and the (AdminService) is administrative. NSMService has both administrative and operational api to manage the namespace. Detailed documentation of each service can be found in the /WEBSVC/doc/ in Solaris.

Most data types are simple and self-explanatory. Users with simple needs can utilize only the operational services, freeing them from the need to manage objects on the Cisco Configuration Engine by means of the AdminService. Advanced users can utilize the AdminService to manage their objects and exercise a more fine-grained control over the operational services.

Managed vs. Un-managed Objects

Operational methods utilize objects, such as Config, Template, Device or Image. These objects are either explicitly supplied in the request, or are created beforehand (using the administrative service) and then referenced by name: the difference is in whether the user manages these objects outside the scope of this request or not. Managed objects are administratively created and/or edited by the user a priori, i.e. before a job is submitted. After the job is complete, the user is responsible for deleting the objects, if desired. Un-managed objects are objects that exist only within the request – the user doesn't have to manage them on the system.

- **Managed Objects**
  - must already exist on the system.
  - are not deleted upon completion of the operation.
- **Un-managed Objects**
  - must *not* already exist on the system
  - must be unique within the request will exist within the system only for the duration of the operation (i.e. are created before any action is taken and deleted immediately after the request completes, successfully or otherwise). If the operation submits a job, the objects exist until the job reaches the COMPLETED, CANCELLED or INVALID state.

For example (from ImageService):

Managed: here, the Device objects are explicitly defined in their entirety.

String *queryInventory*(**Device[] devices**, InventoryJobProperty jobProperty)

Un-managed: here, the devices are simply referenced by their unique name.

String *queryInventoryByDeviceName* (**String[] deviceNames**, InventoryJobProperty jobProperty)

When submitting jobs with un-managed objects, to avoid the possibility of the client request timing out before the server can create all the objects (i.e. if the request was for a very large number of devices) the service will immediately return the job id BEFORE processing the objects (i.e. creating devices, images etc. in the data store, which could take a long time). While processing the objects the job will be in the

PREPARING state, then will become PREPARED and eventually IN_PROGRESS or SCHEDULED. If there is any error preparing the job, the job becomes INVALID. The error message will be contained in the detailed status request for the job, such as getImageDetailStatus().

Aggregate Objects

When the user is managing objects (i.e. devices, configurations and images) by means of the AdminService, only one object can be created/edited/deleted per operation. The system cannot guarantee that the request won't time out (system or network load will affect processing time) thus cannot guarantee returning the result of the operation before the client request times out. When submitting jobs with un-managed objects, aggregate objects *can* be created by means of arrays. The submit operation will *immediately* return a job id before validating or processing the request, thus working around the timeout issue. If any error occurs during processing, the job moves into the INVALID state. Thus, the operational web services do support aggregate objects; however, the AdminService does not.

Error-Handling

Errors during web service operations result in a SOAP fault. Each service has its own error type (or Exception as in the Java paradigm):

- ConfigServiceException
- ImageServiceException
- ExecServiceException
- AdminServiceException
- NSMServiceException

Each exception introduces two properties called serviceCode (3 digits) and statusCode (5 digits), which are used to identify the service and the root cause of this exception. The serviceCode xxx and statusCode yyyyy will be prefixed to the description message in the following format:

[[xxx-yyyyy]] Error detail message…

For example, trying to create an already existing device "router1" with the AdminService would result in the following exception message:

[[001-02001]] DeviceAlreadyExistsException: Device [router1] already exists.

Java clients will simply receive an exception as if it had been thrown locally, whereas non-java users can parse the <faultstring> element to obtain the relevant codes.

The SOAP of an exception is shown below:

```
  <soapenv:Fault>
   <faultcode>soapenv:Server.userException</faultcode>
   <faultstring>com.cisco.netmgmt.ce.websvc.config.ConfigServiceException: [[001-01001]]
Authentication failed: the password is invalid.</faultstring>
   <detail>
    <com.cisco.netmgmt.ce.websvc.config.ConfigServiceException
xsi:type="ns1:ConfigServiceException" xmlns:ns1="http://imgsrv.cns.cisco.com"/>
   </detail>
  </soapenv:Fault>
```

The Definitions of Service Codes

Service Code	Service Name
001	AdminService
002	ConfigService & ExecService
003	ImageService
004	NSMService

Definitions of Status Codes and Their Root Causes

Error Type	Code	Root cause
Generic	01002	InvalidParameterException
	01003	InternalServerException
Device related	02001	DeviceAlreadyExistsException
	02002	DeviceNotFoundException
	02003	DeviceCreationFailsException
	02004	DeviceEditFailsException
	02005	DeviceDeletionFailsException
	02007	SchemaEditFailsException
	02301	InvalidParameterException
	02304	ObjectAlreadyExistsException
	02310	ObjectNotFoundException
	02314	OperationTimedOutException
	02315	OperationFailedException
Group related	03152	MemberAlreadyExistsException
	03153	MemberNotFoundException
	03154	InvalidMemberException
	03155	ParentNotFoundException

	03157	DatastoreAccessException
	03170	InvalidInputException
	03171	CommunicationException
	03172	OperationTimeoutException
	03173	MessageFormatException
Image related	04003	ImageCreationFailsException
	04004	ImageEditFailsException
	04005	ImageDeletionFailsException
Template related	05001	TemplateAlreadyExistsException
	05002	TemplateNotFoundException
	05003	TemplateCreationFailsException
	05004	TemplateEditFailsException
Job related	06001	JobSubmitFailsException
	06002	JobStopFailsException
	06003	JobRestartFailsException
	06004	JobCancelFailsException
	06005	JobDeleteFailsException
	06006	JobListFailsException
	06007	JobNotFoundException
Log related	07002	LogNotFoundException
User related	08001	UserAlreadyExistsException
	08002	UserNotFoundException
	08003	UserCreationFailsException

Conditon related	09002	ConditionNotFoundException
	09003	ConditionCreationFailsException
	09004	ConditionEditFailsException
Query related	10002	QueryNotFoundException
	10003	QueryCreationFailsException
	10004	QueryEditFailsException
	10005	QueryDeletionFailsException
	10006	QueryListFailsException
	10007	QueryRenameFailsException
Notification related	11001	NotificationException
Namespace related	12010	InvalidInputException
	12011	CommunicationException
	12012	OperationTimeoutException
	12013	MessageFormatException
	12200	NamespaceAdminException
	12201	OperationFailedException
	12202	NamespaceAlreadyExistsException
	12203	NamespaceNotFoundException
	12204	MappingAlreadyExistsException
	12206	MappingNotFoundException
	12207	InvalidMappingException
	12208	SubjectAlreadyExistsException
	12209	SubjectNotFoundException
	12211	GroupExistsException
	12212	DatastoreAccessException

Common Semantics

The two main operational services are the ConfigService and ImageService. They share a common semantic for job management:

Task	Operation Name	Example
Submitting jobs	update<type>	updateDevice() updateDeviceWithConfig() updateDeviceWithTempl()
Job lifecycle management	stopJob restartJob cancelJob deleteJob	stopJob() restartJob() cancelJob() deleteJob()
Listing jobs	listJob<type>	listJob() listJobByDevice() listJobWithStatus()
Querying job status	get<jobType>Status get<jobType>DetailStatus	getConfigStatus() getInventoryStatus() getImageDetailStatus() getDeleteFileDetailStatus()

Other methods are specific to the service and are documented in the provided Javadoc.

Security

Security is provided by means of Authentication and Privacy. HTTP Basic Authentication/SSL is the scheme used. For Web Service clients, this will require the use of a Web Services/SOAP toolkit that supports SSL. In the case of a Java client, this can be done natively using JDK 1.4 or greater.

Authentication The credentials of the Cisco Configuration Engine User (created by means of the AdminService) – username and password – are required for each operation. They are passed from client to server by utilizing HTTP Basic Authentication (i.e. the username and password are base-64 encoded into the HTTP header, just as in an HTML form).

For an example of how to embed the credentials (encoded in base-64) in the HTTP headers using client code generated from the Apach AXIS toolkit, see "Setting Credentials into an org.apache.axis.client.Stub" section on page 5-16.

Privacy is provided by SSL and can be enabled during setup on the Cisco Configuration Engine. Only client-side SSL is provided. This means any client can freely obtain an SSL connection to the Config Engine. After the SSL handshake & certificate exchange/acceptance, the communication between the client and server is thus encrypted.

For information on client truststore usage and the "core" cryptographic services defined in the Java 2 SDK, v 1.4, see "JSSE & Keytool Guide" section on page 5-17.

Example Scenarios & Sample Code

For complete code samples of most web service operations, see"Web Services Testing Tool" section on page 5-20.

Scenario 1: Send Configuration to Non-agent-enabled Device (using un-managed objects)

The ConfigService web service is used to submit a config job to a device not running IOS config or exec agents. The device does not already exist on the Cisco Configuration Engine so is defined for this operation only. The config will be associated with the device (by means of the setApplyToDevices() method).

public String **updateConfig**(Config[] configs, ConfigJobProperty jobProperty, Token token)

throws *ConfigServiceException*

Parameters:

configs – an array of Config objects.

jobProperty – a ConfigJobProperty object containing config job specific properties.

token – a Token object (unsupported in this release).

Returns:

A job identifier for this config job. It will be used to query status and delete, cancel, or stop this job.

Throws:

ConfigServiceException if any error occurs, with a corresponding status code and message.

The basic steps are:

1. Prepare config service client stub.
2. Create input parameter objects (Config, Device, HopInfo & ConfigJobProperty) and associate them.
3. Call updateConfig().

Prepare ConfigService Client Stub with Username and Password

Note This sample code uses Apache Axis generated stubs

```
private CEConfig m_svc = null;
CEConfigServiceLocator m_locator = new CEConfigServiceLocator();
m_svc = m_locator.getCEConfigService();

CEConfigServiceSoapBindingStub _stub =
            (CEConfigServiceSoapBindingStub)m_svc;

//set credentials
_stub.setUsername(sUsername);
_stub.setPassword(sPassword);
```

Prepare Parameters

Prepare Config object.

```
// Prepare device objects for config object.
Device[] _devices = new Device[1];

Device _device = new Device();
_device.setName(“router1”);
_device.setType(“NON_AGENT_ENABLED_DEVICE”);

// Non-Agent-enabled device

// hostname of the CE hosting the event gateway
_device.setGatewayId("my-ce-host"); _device.setDeviceType("IOS");

// tells the agent proxy to proxy the config agent
_device.setAgentType("Config Agent");
HopInfo[] _hopInfos = new HopInfo[2]; // device hops for agent proxy
_hopInfos[0] = new HopInfo();
_hopInfos[0].setHopType("IOS_EN");
_hopInfos[0].setIpAddr("10.1.27.44");
_hopInfos[0].setPort(23);
_hopInfos[0].setPassword("cisco");

_hopInfos[1] = new HopInfo();
_hopInfos[1].setHopType("IOS_LOGIN");
_hopInfos[1].setIpAddr("");
_hopInfos[1].setUsername("");
_hopInfos[1].setPassword("cisco");

_device.setHopInfos(_hopInfos); // set hops into device

_devices[0] = _device;

// Config object
Config _config = new Config();
_config.setName("actest_tmp01");
_config.setType("LEGACY");
_config.setLines(_lines); // String[] of cli commands

// associate the Config with the Device
_config.setApplyToDevices(_devices);

Config[] _configs = new Config[1];
_configs[0] = _config;
```

Prepare ConfigJobProperty Object

```
ConfigJobProperty _property = new ConfigJobProperty();
_property.setBatchSize(1);
_property.setSyntaxCheck(true);
_property.setAction("write");
_property.setStartAt(Calendar.getInstance()); // start job now
```

Prepare Token object.

Token object is not supported in Cisco Configuration Engine, therefore null will be passed.

Complete Code

```
/*
 * UpdateConfig.java
 *
 * Copyright (c) 2005 by Cisco Systems, Inc.,
 * 170 West Tasman Drive, San Jose, California, 95134, U.S.A.
 * All rights reserved.
 *
 */

package test;

import java.net.URL;
import java.util.Calendar;
import com.cisco.netmgmt.ce.websvc.common.device.Device;
import com.cisco.netmgmt.ce.websvc.common.device.HopInfo;
import com.cisco.netmgmt.ce.websvc.config.CEConfig;
import com.cisco.netmgmt.ce.websvc.config.CEConfigServiceLocator;
import com.cisco.netmgmt.ce.websvc.config.CEConfigServiceSoapBindingStub;
import com.cisco.netmgmt.ce.websvc.config.Config;
import com.cisco.netmgmt.ce.websvc.config.ConfigJobProperty;
import com.cisco.netmgmt.ce.websvc.config.ConfigServiceException;

public class UpdateConfig
{
    /**
    *  Client stub to service.
    */
    private CEConfig m_svc = null;
    private static final String m_usage =
        "Usage: <numDevices> <devTypeString> <username> <password> [<hostname>]\n" +
        "where devTypeString= AGENT_ENABLED_DEVICE or NON_AGENT_ENABLED_DEVICE";

    private void init(String hostname)
    {
    try
        {
            CEConfigServiceLocator m_locator = new CEConfigServiceLocator();
            System.out.println("hostname: '" + hostname + "'");

            if (hostname == null)
            {
                m_svc = m_locator.getCEConfigService();
                System.out.println("Initialized service! [" +
                m_locator.getCEConfigServiceAddress() + "]");
            }
            else
            {
                String _sSvcNm = m_locator.getCEConfigServiceWSDDServiceName();
                URL _endPoint = new URL(
                "http://" + hostname + "/cns/services/" + _sSvcNm);
                m_svc = m_locator.getCEConfigService(_endPoint);
                System.out.println("Initialized service with URL: " + _endPoint);
            }
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }
    public static void main(String args[])
    {
```

```
    UpdateConfig _et = new UpdateConfig();

    /*
        *  Get command line args...
    */
    String _sNumDevices = null;
    String _sDevType = null;
    String _sHostNm = null;
    String _sUsername = null;
    String _sPassword = null;

    if (args.length < 4)
    {
    p(m_usage);
    System.exit(1);
    }
    _sNumDevices = args[0];
    _sDevType = args[1];
    _sUsername = args[2];
    _sPassword = args[3];

    if (args.length == 5)
    {
    _sHostNm = args[4];
    }

    /*
    *  Initialize service stub...
    */
    _et.init(_sHostNm);

    /*
    *  Set username & password...
    */
    _et.setCredentials(_sUsername, _sPassword);

    /*
        *  Invoke service...
    */
    _et.execute(Integer.parseInt(_sNumDevices), _sDevType);
}

public void execute(int numberOfDevices, String type)
{
    try
    {
        Device[] _devices = new Device[numberOfDevices];

        for (int i = 0; i < numberOfDevices; i++)
        {
            Device _device = new Device();
            _device.setName("router1");
            _device.setType("NON_AGENT_ENABLED_DEVICE");

            // Non-Agent-enabled device
            // hostname of the CE hosting the event gateway
            _device.setGatewayId("my-ce-host");
            _device.setDeviceType("IOS");
            // tells the agent proxy to proxy the config agent
            _device.setAgentType("Config Agent");

            HopInfo[] _hopInfos = new HopInfo[2];
            _hopInfos[0] = new HopInfo();
            _hopInfos[0].setHopType("IOS_EN");
```

```
                _hopInfos[0].setIpAddr("10.1.27.44");
                _hopInfos[0].setPort(23);
                _hopInfos[0].setPassword("cisco");

                _hopInfos[1] = new HopInfo();
                _hopInfos[1].setHopType("IOS_LOGIN");
                _hopInfos[1].setIpAddr("");
                _hopInfos[1].setUsername("");
                _hopInfos[1].setPassword("cisco");

                _device.setHopInfos(_hopInfos); // set hops into device

                _devices[i] = _device;
            }

            ConfigJobProperty _property = new ConfigJobProperty();
            _property.setBatchSize(1);
            _property.setSyntaxCheck(true);
            _property.setAction("write");
            _property.setDescription("Submit through WEB SERVICE API.");
            _property.setStartAt(Calendar.getInstance());

            String[] _lines = new String[1];
            _lines[0] = "ip host alex-rm3 10.1.1.1";

                Config _config = new Config();
                _config.setName("actest_tmp01");
                _config.setType("LEGACY");
                _config.setLines(_lines);


            // associate the Config with the Device
            _config.setApplyToDevices(_devices);

            Config[] _configs = new Config[1];
            _configs[0] = _config;

            // the actual call to the server
            String _jobId = m_svc.updateConfig(_configs, _property, null);

            System.out.println("Job id: " + _jobId);
        }
        catch(ConfigServiceException ex)

        {
            ex.printStackTrace();
            System.out.println("Error Code: " + ex.getErrorCode());
            System.out.println("Msg: " + ex.getMessage());
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
    }

    public void setCredentials(String sUsername, String sPassword)
    {
        CEConfigServiceSoapBindingStub _stub =
        (CEConfigServiceSoapBindingStub)m_svc;
        _stub.setUsername(sUsername);
        _stub.setPassword(sPassword);
        p("Credentials set.");
    }
```

```
    private static void p(Object o)
    {
    System.out.println(o);
    }
}
```

Scenario 2: Create Agent-enabled Device in Default Group (managed object)

This creates a device in the default group "/config/default." This device can be referenced by name in any Web Service operation with an input parameter asking for a String deviceName.

This is an example of a managed object since the user is explicitly creating and managing this device by means of the AdminService, outside the scope of the operational web services ConfigService, ImageService and ExecService.

public int create**DeviceWithAttr**(CNSDevice device, CNSAttribute[] attrs, Token token)

Throws *AdminServiceException*

A public method to create a device and put in default group.

Parameters:

device - the CNSDevice that contains the device attributes.

attrs - the additional attributes and/or overriding attributes to set.

token - a Token object.

Returns:

int success = 0, failure = anything else

Throws:

AdminServiceException

java.rmi.RemoteException

Before using createDevice API, there are few setups that need to be done, prepare admin service and API parameters (complete code is provided at the end of this section).

Prepare Admin Service with Username and Password

```
private CEAdmin m_svc = null;

CEAdminServiceLocator m_locator = new CEAdminServiceLocator();
m_svc = m_locator.getCEAdminService();

//set credential
CEAdminServiceSoapBindingStub _stub =
            (CEAdminServiceSoapBindingStub)m_svc;

_stub.setUsername(sUsername);
stub.setPassword(sPassword);
```

Prepare Parameters

Prepare CNSDevice Object

```
// Prepare device objects for config object.

     CSNDevice _cnsDevice = new CNSDevice();
     _cnsDevice.setDeviceName(“myCNSDevice”);
     _cnsDevice.setId(“myCNSDevice”);
```

Prepare Attributes

```
   // Attribute 1
   CNSAttribute _attr1 = new CNSAttribute();
   _attr1.setName("IOShostname");

   String[] _values = new String[1];
   _values[0] = "myhost ";

   _attr1.setValues(_values);

   // Attribute 2
   CNSAttribute _attr2 = new CNSAttribute();
  _attr2.setName("IOSdomain");

   _values = new String[1];
   _values[0] = "mycom.com";

   _attr2.setValues(_values);

   // Attribute 3
   CNSAttribute _attr3 = new CNSAttribute();
   _attr2.setName("IOSpassword");

   _values = new String[1];
   _values[0] = "mypass";

   _attr3.setValues(_values);


   CNSAttributeMsg[] _attrs = new CNSAttributeMsg[3];
   _attrs[0] = _attr1;
   _attrs[1] = _attr2;
   _attrs[2] = _attr2;
```

Prepare Token Object

Token object is not supported in this version, therefore null will be passed to the API.

Complete Code

```
/*
 * CreateDeviceWithAttr.java
 *
 * Copyright (c) 2005 by Cisco Systems, Inc.,
 * 170 West Tasman Drive, San Jose, California, 95134, U.S.A.
```

```
 * All rights reserved.
 *
 */

import com.cisco.netmgmt.ce.websvc.admin.*;
import com.cisco.netmgmt.ce.websvc.admin.CNSAttribute;
import java.rmi.RemoteException;

public class CreateDeviceWithAttr
{
    /**
     *  Client stub to service.
     */
    private CEAdmin m_svc = null;

    private void init(String hostname)
    {
        try
        {
            CEAdminServiceLocator m_locator = new CEAdminServiceLocator();

            if (hostname == null)
            {
                m_svc = m_locator.getCEAdminService();
                System.out.println("Initialized service! [" +
                    m_locator.getCEAdminServiceAddress() + "]");
            }
            else
            {
                String _sSvcNm = m_locator.getCEAdminServiceWSDDServiceName();
                URL _endPoint = new URL(
                    "http://" + hostname + "/cns/services/" + _sSvcNm);
                m_svc = m_locator.getCEAdminService(_endPoint);
}
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }


public class TestCreateDeviceWithAttr
    extends CEAdminService
{
    public static void main(String args[])
        throws Exception
    {
        CreateDeviceWithAttr _ti = new CreateDeviceWithAttr ();
        _ti.init(myhost);

        String _devName = args[0];
        _ti.execute(_devName);
    }

    public void execute(String devName)
        throws RemoteException
    {
        CNSDevice _cnsDev = new CNSDevice();
        _cnsDev.setDeviceName(devName);
        _cnsDev.setId(devName);

        // Attribute 1
        //CNSAttribute _attr1 = new CNSAttribute();
```

```
        CNSAttributeMsg _attr1 = new CNSAttributeMsg();
        _attr1.setName("IOShostname");

        String[] _values = new String[1];
        _values[0] = "ios_slinky";

        _attr1.setValues(_values);

        // Attribute 1
        //CNSAttribute _attr2 = new CNSAttribute();
        CNSAttributeMsg _attr2 = new CNSAttributeMsg();
        _attr2.setName("IOSdomain");

        _values = new String[1];
        _values[0] = "cisco.com";

        _attr2.setValues(_values);

        // Attribute 1
        //CNSAttribute _attr3 = new CNSAttribute();
        CNSAttributeMsg _attr3 = new CNSAttributeMsg();
        _attr2.setName("IOSpassword");

        _values = new String[1];
        _values[0] = "blender";

        _attr3.setValues(_values);


        CNSAttributeMsg[] _attrs = new CNSAttributeMsg[3];
        _attrs[0] = _attr1;
        _attrs[1] = _attr2;
        _attrs[2] = _attr2;

        try
        {
            m_svc.createDeviceWithAttr(_cnsDev, _attrs, null);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Notes

Setting Credentials into an org.apache.axis.client.Stub

This document provides information for clients using the Apache Axis 1.1 or later as a client on how to set credentials (username & password) into a class extending **org.apache.axis.client.Stub**. The credentials are set using the HTTP Basic Authentication scheme. An example header: Authorization: Basic YWRtaW46Y2lzY28=

This document assumes a working knowledge of the Apache Axis SOAP Toolkit for Java, and that you have already generated your Java web service client stubs using WSDL2Java.
See **http://ws.apache.org/axis/java/user-guide.html#WSDL2Java**.

The steps to initialize a web service stub, set the credentials into the stub and invoke a service method are shown below. In this example, the service name is Cisco Configuration Engine ConfigService:

```
// get service interface
CEConfigServiceLocator _locator = new CEConfigServiceLocator();
CEConfig _svc = _locator.getCEConfigService(_endPointURL);

// cast to binding implementation of stub and set credentials
CEConfigServiceSoapBindingStub _stub =
    (CEConfigServiceSoapBindingStub)_svc;
_stub.setUsername(sUsername);
_stub.setPassword(sPassword);

// invoke a service method...
_svc.someMethod(someArg);
```

Note This might not be the only method to achieve this, but it is one that is tested. Also note it presumes knowledge of the transport, which Web Services hides.

JSSE & Keytool Guide

This document provides information on the following topics:

- JSSE
- Troubleshooting Tips
- Keystores and Truststores
- Configuration Requirements (JSSE Sample Code)
- Creating a simple keystore and truststore (Unix)
- “keytool” Java Utility
- OpenSSL

JSSE

JSSE provides both an application programming interface (API) framework and an implementation of that API. The JSSE API supplements the *core* cryptographic services defined in the Java 2 SDK, v 1.4, java.security and java.net packages by providing extended networking socket classes, trust managers, key managers, SSLContexts, and a socket factory framework for encapsulating socket creation behavior.

The JSSE implementation in the J2SDK, v 1.4 implements SSL 3.5 and TLS 1.0. It does not implement SSL 2.0.

- JSSE Standard API
- javax.net
- javax.net.ssl
- javax.security.cert

Using https:// on JDK 1.4.2 transparently enables SSL-enabled sockets.

Troubleshooting Tips

Problem:

javax.net.ssl.SSLHandshakeException: unknown certificate Cause: server certificate not present in the client truststore, or the jvm is using a different truststore than you expect.

Solution:

- Check the client truststore to determine if the signer certificate from the server personal certificate is there. For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate. For a CA signed server personal certificate, the signer certificate is the root CA certificate of the CA which signed the personal certificate.
- Add the server signer certificate to the client truststore.
- To indicate to the jvm where the truststore is located, provide the following argument:

  -Djavax.net.ssl.trustStore=$TRUSTSTORE_ABS_PATH

  Example:

  TRUSTSTORE_ABS_PATH=/users/boniface/.keystore

background information

http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

Keystores and Truststores

A keystore is a database of key material. Key material is used for a variety of purposes, including authentication and data integrity. There are various types of keystores available, including PKCS12 and Sun's JKS.

Generally speaking, keystore information can be grouped into two different categories:

- key entries
- trusted certificate entries.

A key entry consists of an entity's identity and its private key, and can be used for a variety of cryptographic purposes. In contrast, a trusted certificate entry only contains a public key in addition to the entity's identity. Thus, a trusted certificate entry cannot be used where a private key is required, such as in a **javax.net.ssl.KeyManager**. In the J2SDK implementation of JKS, a keystore might contain both key entries and trusted certificate entries.

A truststore is a keystore which is used when making decisions about what to trust. If you receive some data from an entity that you already trust, and if you can verify that the entity is the one it claims to be, then you can assume that the data really came from that entity.

An entry should only be added to a truststore if the user makes a decision to trust that entity. By either generating a keypair or by importing a certificate, the user has given trust to that entry, and thus any entry in the keystore is considered a trusted entry.

It might be useful to have two different keystore files: one containing just your key entries, and the other containing your trusted certificate entries, including Certification Authority (CA) certificates. The former contains private information, while the latter does not. Using two different files instead of a single keystore file provides for a cleaner separation of the logical distinction between your own certificates (and corresponding private keys) and others' certificates. You could provide more protection for your private keys if you store them in a keystore with restricted access, while providing the trusted certificates in a more publicly accessible keystore if needed.

Configuration Requirements (JSSE Sample Code)

When running the sample programs that create a secure socket connection between a client and a server, you will need to make the appropriate certificates file (truststore) available. For both the client and the server programs, you should use the certificates file *samplecacerts* from the samples directory. Using this certificates file will allow the client to authenticate the server. The file contains all the common Certification Authority certificates shipped with the J2SDK (in the *cacerts* file), plus a certificate for *duke* needed by the client to authenticate *duke* when communicating with the sample server ClassFileServer. (ClassFileServer uses a keystore containing the private key for *duke* which corresponds to the public key in samplecacerts.)

To make the samplecacerts file available to both the client and the server, you can either copy it to the file *<java-home>/lib/security/jssecacerts*, rename it cacerts and use it to replace the *<java-home>/lib/security/cacerts* file, or add the following option to the command line when running the java command for both the client and the server:

-Djavax.net.ssl.trustStore=path_to_samplecacerts_file

If you use a browser, such as Netscape Navigator or Microsoft's Internet Explorer, to access the sample SSL server provided in the ClassFileServer example, a dialog box might pop up with the message that it does not recognize the certificate. This is normal because the certificate used with the sample programs is self-signed and is for testing only. You can accept the certificate for the current session. After testing the SSL server, you should exit the browser, which deletes the test certificate from the browser's namespace.

Creating Simple Keystore and Truststore (Unix)

http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html#CreateKeystore

Create a simple JKS keystore suitable for use with JSSE with a trusted certificate entry. All keystore entries are accessed by means of unique, case-insensitive aliases.

```
Keystore: ~/.keystore   [this is the default]
Alias:    config_engine
Password: *****
Certificate imported: ~/wd/auth/ce.base64.cer
```

Java Utility keytool

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html#importCmd

Usage:

- IMPORT a certificate (creates new alias w/ name given & keystore ~/.keystore):

  keytool -import -alias config_engine -file ~/wd/auth/ce.base64.cer
  (you will be prompted for a password: remember if creating a new keystore).

- LIST certificates in keystore (identified by alias):

  keytool -list [-alias config_engine]

- CREATE a new keystore and self-signed certificate with corresponding public/private keys:

  keytool -genkey -alias config_engine1 -keyalg RSA -validity 3650 -dname "CN=boniface, OU=nmtg, O=cisco, L=san jose, S=ca, C=us"

- EXPORT certificate in keystore (identified by alias) [rfc means base64 format]

  keytool -export -alias config_engine1 -rfc -v -file config_engine1.base64.cer

- DELETE alias

keytool -delete -alias xyz

OpenSSL

http://www.openssl.org/

[Documents --> HOWTO]

http://www.openssl.org/docs/HOWTO/certificates.txt

Relationship between certificates & keys:

Certificates are related to public key cryptography by containing a public key. To be useful, there must be a corresponding private key somewhere. With OpenSSL, public keys are easily derived from private keys, so before you create a certificate or a certificate request, you need to create a private key.

http://www.openssl.org/docs/HOWTO/keys.txt

Private keys are generated with *openssl genrsa* if you want a RSA private key.

Web Services Testing Tool

Step 1 Open *src/AdminConditionTests.xml*, replace **admin.TestGetConditions1** with **admin.TestGetConditions**.

Step 2 Open *src/AdminTests.xml*, remove suite **ListAllAttributeNames**.

Step 3 Open *src/ConfigTests.xml*, replace **config.TestDeleteConfigJob1** with **config.TestDeleteConfigJob**, and replace **config.TestCancelConfigJob1** with **config.TestCancelConfigJob**.

Step 4 Open *src/ImageTests.xml*, replace **admin.TestEditImageLocation1** with **admin.TestEditImageLocation**, and **admin.TestSetImageId1** with **admin.TestSetImageId**.

Depreciated Image Web Service

Note These examples have not been updated for Cisco Configuration Engine because this service has been depreciated. All users are STRONGLY encouraged to utilize the newer ImageService. This service is only provided for backward compatibility with specific customers.

The Image Web Service is a Web Services interface to the Image Service component of the Cisco Configuration Engine. The Image Service provides image management for IOS and non-IOS devices on a large scale.

The Image Web Service provides remote, programmatic access to the Image Service. It is simply a Web Services interface to the Image Service and does not extend the functionality of the Image Service itself.

This feature enables users to deploy images on a mass scale in an automated fashion using a single management point, whereby the user can be a human interfacing through a Web-browser or a program interfacing through the Web Service interface.

With a single operation, you can trigger an intelligent image download to multiple devices whereby the images can reside within or external to the domain of a server. This server operates with the Image Agent imbedded in IOS to provide added value over the usual copy flash command, such as pre-validation of the image against the destination hardware resources before commit of the image into the destination device.

The Image Web Service provides the following capabilities:

- Query inventory (file-systems, contents, running image version, and so forth).
- Distribute image files to a specific location on a device.
- Activate images at a specific location on a device.
- Submit image distributions and activations in batch mode (a job) to one or *n* devices.
- Manage a job (create, stop, cancel, restart, list, or query the status of one or more jobs).

The functionality provided by the Image Service can be categorized into two main types:

- Administrative functions – creating and managing device and image objects as well as the associations between them.
- Operational functions – obtaining device inventory requests; triggering distribution, activation, and evaluation jobs; querying the status of jobs.
- The Cisco Configuration Engine interacts with devices using the Event Bus.
- Devices can initiate HTTP(S) connections with the Cisco Configuration Engine.
- Devices download their images from an image repository over (T)FTP.

End User Interface

The following primitives define the API and its semantics. These primitives assume no specific implementation technology (Corba IDL, WSDL, RMI, and so forth). Most data types are self-describing.

Exception Handling

As with any distributed programming protocol, there are exceptions relating to the transport and exceptions relating to the application. The former are specific to the type of Web Service toolkit or client environment used to invoke the Web Service.

In the case of a Java client developed using the Apache AXIS toolkit, all service methods implicitly throw a *java.rmi.RemoteException*. They have nothing to do with the application's functionality but rather are errors in transporting the SOAP from the client across the network to the server hosting the Web Service. They will be specific to the client environment.

Again, referring to an Apache AXIS Java client, all Image Web Service methods can report these general transport-related exceptions that are mapped into a SOAP Fault, not described in the WSDL. The *faultcode* of this contains the classname of the fault. The recipient is expected to deserialize the body of the fault against the classname.

All application exceptions are expressed through a single exception message type: *com.cisco.cns.imgsrv.CISException*. It is also mapped to a SOAP Fault, but it in this case it is described in the WSDL as a <wsdl:message> element.

Exception handling in Web Services is an area of some confusion. For more information, see Section 5.5.5 and Chapter 14 in the JAX-RPC specification. For AXIS-specific information, see the Exception Handling section in *http://ws.apache.org/axis/java/user-guide.html*.

Operational Methods

Use Operation Methods to:

- Obtain device inventory requests.
- Trigger distribution and activation.
- Evaluate jobs.
- Query status of jobs.

getImageInventoryReport

Usage:

CnsMessage[] getImageInventoryReport(String[] imageIds)

Description:

Retrieves running image, hardware, and file system information from the specified device(s). You can use this information to make decisions on what distribution or activation scenarios to perform.

Parameters:

imageIds—an array of image IDs that identify the target devices to query.

Returns:

CnsMessage—A tree structure of value-objects with information on each device's running image, hardware resources, and file systems and contents.

Exception Handling:

CISException—if **imageIds** is null, or any array element is invalid (i.e. null, empty string, or white space only), or if there is a server error processing the request.

submitJob

Usage:

SubmitStatus submitJob(JobDTO job)

Description:

Creates an image update Job for a specified list of devices and groups of devices. Devices, when created, have associated distributions of one or more images and the activation of an image present on the device: this job is the enactment of these. The JobDTO job parameter specifies:

- Time this job is scheduled to begin.
- Batch size (number of devices to concurrently execute).
- Optional text description of the job.
- Whether to enact the distribution(s) and activation, or simply evaluate them.

Parameters:

JobDTO job—Defines parameters of the job to be submitted including: start time, batch size, job type, description, and type (distribution and activation, or evaluation). If the start time is later (on the server) than when this method is invoked, the job will be queued and will trigger at that specified time. Otherwise, the job executes immediately.

Returns:

SubmitStatus—Object describing the status of the job request. A submit can have multiple return outcomes, so this assembler handles those different cases.

For example, if any device in this job is already active in another job, the submit fails and a list of the **imageId:jobId** pairs will be returned as *ImageIdJobId[]*.

If all devices are not involved in any prior submitted job, the submit is successful and a unique identifier of this submitted job (**jobId**) for this submitted job as well as the resulting list of any images slated for distribution and activation for each device specified will be returned. The **jobId** can then be used to query the status of a job (simple or detailed status), stop, cancel, or restart the job.

Exception Handling:

CISException if job is null, batch size is < 1, no devices are specified or all groups contain no devices, or any other system error.

evaluateJob

Usage:

SubmitStatus evaluateJob(JobDTO job)

Description:

Submits a job to the server only for evaluation purposes, not for actual execution of the job.

The server evaluates the distributions associated with each device and reports in each case:

- Whether the device is reachable.
- If each distribution is required (i.e. device could already be running the desired image or the file is already present at the destination location).
- If there is sufficient space for the image file at the intended destination location.

If the device has an associated activation, the server also evaluates whether the activation of an image is required (i.e. is the device already running the desired image). This is useful to run an image update scenario and see what resulting actions the server would take, but without actually executing those actions.

Parameters:

JobDTO job—Defines parameters of the evaluation to be submitted, such as start time, batch size, job type, and description.

Returns:

SubmitStatus—Object describing the status of the job request (see **submitJob()** for more detail). Includes a list of which images are being evaluated for distribution and activation for each device, but only if all of the devices are not already members of another prior submitted job.

Exception Handling:

CISException if job is null, batch size is < 1, no devices are specified or all groups contain no devices, or any other system error.

listJobIds

Usage:

String[] listJobIds()

Description:

Get a list of all jobs on the server. If there are no jobs an empty String array is returned.

Returns:

String[]—array containing IDs of all jobs regardless of state.

getJobStatus

Usage:

String getJobStatus(String jobId)

Description:

Get the status of a job as a single String, one of:

- CANCELLED
- COMPLETED
- IN_PROGRESS
- SCHEDULED
- STOPPED
- STOPPING

Parameters:

String jobId—The ID of the job for which to query the status.

Returns:

String— job status.

JobStatusDTO—an object defining the job status in detail including:

- Time job was started.
- Job type (distribution, activation, both, or evaluation only).
- If device is reachable.
- If there is enough room to distribute the image to the desired target location on the device's file system.
- If activation is necessary.

Exception Handling:

CISException if jobID is invalid, i.e. no job for the given **jobID** can be found.

getJobDetailStatus

Usage:

JobStatusDTO getJobDetailStatus(String jobID)

Description:

Get detail status for this job.

Parameters:

String jobId—The ID of the job for which to query the detail status.

Returns:

JobStatusDTO—an object defining the job status in detail including:

- Time job was started.
- Job type (distribution, activation, both, or evaluation only).
- If device is reachable.
- If there is enough room to distribute the image to the desired target location on the device's file system.
- If activation is necessary.

Exception Handling:

CISException if jobID is invalid, i.e. no job for the given **jobID** can be found.

stopJob

Usage:

void stopJob(String jobID)

Description:

Places the job identified by the **jobID** into the STOPPED state.

Parameters:

jobId—ID of job you wish to stop.

Exception Handling:

CISException if jobID is invalid, i.e. no job for the given **jobID** can be found.

restartJob

Usage:

void restartJob(String jobID)

Description:

Moves a job from the STOPPED state into the IN_PROGRESS state if possible, restarting the job. Jobs (distributions and activations) are idempotent so repeating steps is not harmful.

Parameters:

jobId—ID of job you wish to restart.

Exception Handling:

CISException if jobID is invalid, i.e. no job for the given **jobID** can be found.

cancelJob

Usage:

void cancelJob(String jobID)

Description:

Cancel a previously submitted job.

Parameters:

jobId—ID of job you wish to cancel.

Exception Handling:

CISException if jobID is invalid, i.e. no job for the given **jobID** can be found.

Administrative Methods

These methods are administrative methods only. The **bulkUpload()** method creates application objects such as:

- Devices
- Images
- Groups

There is currently no means to modify objects exclusively through the Web Service (though they can be modified through the web browser interface to the Cisco Configuration Engine). They must be deleted and recreated with the new values. Also, devices cannot be added to an existing group, only to new groups defined within the same XML document.

deleteDevice

Usage:

deleteDevice(String imgId)

Description:

Deletes a device along with its Configuration Service and Image Service attributes. It does not delete any associated images because these can be used by other devices in the system.

Parameters:

imgId—identifier of the device to be deleted. This ID maps to the <cns-extended-attr name="IOSConfigID"> and <image-id> child elements of a <cns-device-info> element when defining a device in a **bulkupload()** XML file argument. For more information, see **bulkupload()** method.

Note The method deleteImgwDevice(Srting id) is now obsolete. Now, there is no separate IMGW device. Devices are a single entity and are either agent-enabled, nonagent-enabled, or PIX devices. See the *Cisco Configuration Engine Administration Guide*.

deleteGroup

Usage:

deleteGroup

Description:

Deletes a group.

Parameters:

id—identifier of the device to be deleted. This maps to the <cns-group-name> child element of the <cns-group-info> element when defining a group in a **bulkupload()** XML file argument.

Exception Handling:

CISException—If any system error occurs processing the request.

deleteImage

Usage:

void deleteImage(String imgName)

Description:

Deletes an image object.

Parameters:

imgName—name of the image to be deleted. This maps to the <name> child element of the <image> element when defining an image in a **bulkupload()** XML file argument.

Exception Handling:

CISException—If any system error occurs processing the request.

Import Template

Usage:

int importTemplate (int protocol, String serverName, String user, String password, String remoteDirectory, String fileName) .

Description:

Imports a template file from remote machine to the Config Server.

Note The Cisco Configuration Engine 3.5.3 supports only single file import at a time in bulk upload.

Parameters:

- *protocol—the* protocol to be used for import of template (FTP=0, SFTP=1).
- *serverName—the* remote machine from where the file has to be imported.
- *user—the* username to login into the remote server.
- *password*—password to login to the remote server.
- *remoteDirectory*—absolute path of the directory in the remote machine from where the file has to be imported.
- *fileName*—name of the file to be imported.

Returns:

Returns an integer value

0	Success
Any other value	Failure

Exception Handling:

- AdminServiceException if there is an error in importing templates.
- RemoteException if there is an error communicating with the service.

Export Template

Usage:

int exportTemplate (int protocol, String serverName, String user, String password, String remoteDirectory, String fileName) .

Description:

Exports an existing template file from the Config Server to the remote machine.

Note The Cisco Configuration Engine 3.5.3 supports only single file export at a time in bulk upload.

Parameters:

- *protocol—the* protocol to be used for export of template (FTP=0, SFTP=1).
- *serverName—the* remote machine from where the file has to be exported.
- *user—the* username to login into the remote server.
- *password*—password to login to the remote server.
- *remoteDirectory*—absolute path of the directory in the remote machine from where the file has to be exported.
- *fileName*—name of the file to be exported.

Returns:

Returns an integer value

0	Success
Any other value	Failure

Exception Handling:

- AdminServiceException if there is an error in exporting templates.
- RemoteException if there is an error communicating with the service.

bulkUpload

Usage:

String bulkUpload(String xml)

Description:

Uploads object data to the Cisco Configuration Engine Device, Group, Image and Application objects can be defined in XML and uploaded. For detailed information on this feature, see the *Cisco Configuration Engine Administration Guide*.

With Cisco Configuration Engine, objects can be edited or deleted by means of the Bulk Upload tool.

Note Only one bulkUpload operation is permitted to execute at any time. If you attempt to execute two or more concurrent bulkupload requests, all but the first received are ignored, are not queued, and must be resubmitted. A *CISException* with a warning message is returned for all ignored requests.

Parameters:

xml—the bulkupload XML file. See the DTD for the correct format.

Returns:

String – a success message.

Exception Handling:

CISException—If a bulkupload request is already in progress or any system error occurs processing the request. For more information, see the *Cisco Configuration Engine Administration Guide*.

Types

To ensure maximum interoperability, the types used are only the standard XML Schema (XSD) simple types. All are nillable and support the ID and HREF attributes, so in an RPC-encoded context (such as the Image Web Service) multi-ref serialization (for arrays) is supported.

Any custom types (CnsMessage) are XSD complex types that are only comprised of XSD simple types or other complex types. They are essentially like Javabeans; simply an object with properties.

In the following example, the XML schema type JobDTO is actually a bean with properties that are each an XSD simple type, like boolean, string, dateTime. Nesting is supported, so a complex type can contain property whose type is yet another complex type.

```
<multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:JobDTO"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://job.websvc.imgsrv.cns.cisco.com">
<activation xsi:type="xsd:boolean">true</activation>
<batchSize xsi:type="xsd:int">1</batchSize>
<date xsi:type="xsd:dateTime">2003-12-08T04:42:49.105Z</date>
```

```
<description xsi:type="xsd:string">web svc dist &amp; act
evaluation</description>
<distribution xsi:type="xsd:boolean">true</distribution>
<evalutaion xsi:type="xsd:boolean">true</evalutaion>
<imageIds xsi:type="xsd:string">c7200-2</imageIds>
</multiRef>
```

Any SOAP toolkit should be able to auto-map these into an object or stub appropriate for use in the client domain. In the Java example, **xsi:type="ns2:JobDTO"** maps to the **Javabean JobDTO.java** with the following properties:

```
boolean distribution;
boolean activation;
boolean evaluation;
int batchSize;
Calendar date;
String description;
String[] imageIds;
```

Notes

- For users of the EFT version of this Web service, note that groups are now hierarchical. For example, the group *default* in earlier releases of Cisco Configuration Engine is now addressed as */config/default*.
- Device configuration templates must be hosted on a Web server. Configuration templates (used only if a configuration change is needed when activating a new image) for devices must be hosted on a Web server accessible over HTTP from the server.
- When stopping jobs, note that the server can only prevent the next queued operations from being submitted, but might not be able to cancel the currently executing operation for each of the devices in the job. For example, once a device reload is in progress, there is no way to terminate the reload and the server is helpless to prevent the impending activation. However, if during a job that involves a distribution and activation, if the stop is submitted during the distribution phase, the activation can be prevented.

Example Scenarios

Note These examples have not been updated for Cisco Configuration Engine because this service has been depreciated. All users are STRONGLY encouraged to utilize the newer ImageService. This service is only provided for backward compatibility with specific customers.

A scenario that exercises all Web Service methods is described below, from creating devices and image associations to submitting distribution jobs and querying their status, etc. The names used are only for the purposes of this example. They are referenced in some sample bulkupload XML files which are also provided.

Background

Data Management

Application objects – like devices, images, etc. - must be created administratively through the bulkUpload(String xml) web service method. This takes an XML file that describes the objects and persists them for use during the operational methods. Please note that data can be created or deleted through the web service, but not modified.

Objects

IMAGE – defines the attributes of an image file, such as its (t)ftp location(s), MD5, size, name, etc.

DEVICE – defines a router that we wish to distribute and activate image files to. Includes a unique identifier (image-id... I know you'd expect it to be called device-id, but since a device object may utilize multiple services on the server, it's identifier is always relative to the service in question), configurations (in what are called "templates" - basically CLI files with optional parameters which we will be ignoring for now)

IMGW DEVICE – defines the information necessary to access a device through the Intelligent Modular Gateway (IMGW). Since not all devices that the Image Service manages will have or have enabled the agents that control image and configuration management, a legacy communication protocol must be used, such as telnet. The IMGW essentially proxies Event Bus communication to the device.

GROUP – a logical grouping of DEVICE objects.

Scenario One

Objective

Distribute an image (named 7200-27) to a device and activate it.

Assumptions

- All device configuration templates must be hosted on a web server.
- The image file is available on a public FTP directory.
- You have generated client stubs (using your favorite SOAP toolkit for your desired language and platform) from the WSDL with which to invoke the Web Service.

Preconditions

- Device is NOT already running the image 7200-27.
- Device IS running the image 7200-21.

Steps

Note Steps 1 through 3 are ADMINISTRATIVE, creating data objects on the server.

Step 1 First, we need to define on the server the image file by describing the name, location and attributes of the image file residing on the FTP server. Create an image object called **7200-27** (and **7200-21** for use later) through the Web Service method **bulkUpload()** which uploads an XML file describing this object.

Step 2 Using **bulkUpload()** again, create a device object **c7200-2** that is associated with the image **7200-27** inside the group **test** that specifies the activation configuration template *activation.7200-27.cfgtpl*.

Step 3 If the device does not have any agents (or they are not enabled) you can create an IMGW Device (a.k.a. agent proxy) to proxy communication with the device. You would bulkupload an IMGW device named **c7200-2**, the same as the device object.

Note The remaining steps are OPERATIONAL.

Step 4 Submit an image inventory request.

This allows you to see what is currently running on the device, the contents of the file systems, and other information. Invoke the getImageInventoryReports() method indicating which device to get the report from (a String[] array with image-id {"c7200-2"}, the image-id of the device).

Step 5 Submit a job to the server asking it to distribute and activate the image **7200-27** on the device **c7200-2**:

a. Define the job with a **JobDTO** object, specifying the batch-size (since only one device, just specify one).

b. Define an optional job description, where:

activate = **true**

distribute = **true**

evaluate = **false**

device image-id = **c7200-2**

c. Invoke the **submitJob(JobDTO)** method.

Note from the retuned **SubmitStatus** object the job identifier. You will need this to query the status of the job later. Also, the **SubmitStatus** indicates what images have been submitted for distribution or activation for each device.

Note The activation CLI template specifies that the image to be activated is 7200-27.

Step 6 Alternately, you could submit an evaluation (where the server checks if the distribution(s) or activation(s) are required and possible for each device, but doesn't actually perform any of these actions). The steps are identical, except specify evaluation = **true** in the **JobDTO** object and invoke the method **evaluateJob(JobDTO)** (instead of **submitJob**).

Step 7 Now that you have submitted a job (or evaluation) you can periodically check on the status of the job by querying its status through the two status methods:

- **String getJobStatus(String jobID**)

  Returns a simple String description of the job:

  CANCELLED

  COMPLETED

  IN_PROGRESS

  SCHEDULED

STOPPED

STOPPING

- **JobStatusDTO getJobDetailStatus(String jobID)**

  Returns a **JobStatusDTO** that provides substantially more detailed information about the job, such as what time it was started, the job type (distribution, activation, both, or evaluation only), if the device is reachable, if there is enough room to distribute the image to the desired target location on the device's file system, if the activation is necessary, etc.

Continue to query the device until you see the desired COMPLETED status. If an error occurs, it is reported.

Step 8 You can at any time list the IDs of all jobs and evaluations through the **listJobIds**() method.

Any jobs that have previously been cancelled (through the **cancelJob(id)** method) will have been destroyed and thus will not be listed.

Step 9 You can choose at any time to stop an executing job with **stopJob(id)**, restart a stopped job with **restartJob(id)**, or cancel a job with **cancelJob(id)**.

Notes

- Once a job has been cancelled, it is destroyed and no record of it exists any longer (except in the server log files). Distributions, activations, and evaluations are all idempotent thus can all be safely restarted without any ill effects.
- If an image has already been successfully distributed to a device and is still present, the server will not do so again.
- If a device is already running the activated image, then the server will not tell the device to reload that image.

Scenario Two

Objective

Return the device to its original state by distributing the image **7200-21** and activating it.

Assumptions

- All device configuration templates are hosted on a web server.
- The image file is available on a public FTP directory.

Preconditions

- Device is NOT already running the image **7200-21**.
- Device IS running image **7200-27** (from Scenario One).
- Scenario One has successfully completed.

Steps

Currently, there is no way to modify the devices or their associated images (this limitation will be addressed in a future release). To submit new jobs for the same device you must delete then recreate the device and its group. This is a temporary workaround until sufficient data management APIs can be developed.

Step 1 Delete the device **c7200-2** and its group with:

deleteDevice(c7200-2)

deleteGroup(test)

Note It is NOT necessary to delete and recreate the IMGW device unless you are changing its connection information. It is also not necessary to delete and recreate the image objects unless they must be changed.

Step 2 Use bulkupload to recreate the device **c7200-2** that associates the device with image **7200-21** instead of **7200-27**.

Note The activation CLI template (delivered during an activation) specifies that the image to be activated as **7200-21**.

Step 3 Optionally, submit an image inventory request as before to verify the device's running image now.

Step 4 As in of Scenario One, submit a distribution and activation job.

The returned **SubmitStatus** should indicate that image **7200-21** is to be activated and distributed.

The returned job ID is for querying the job later.

Step 5 Query the job to determine its status.

Sample Bulkupload Files

The following sample bulkupload files define the objects referenced in the scenarios above.

Distribute and Activate 7200-21

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-bulk-upload SYSTEM "BulkUpload.dtd">
<cns-bulk-upload stop-on-error="false">
    <cns-element-data>
        <NSM-DATA op-type="add" validate-data="true">
            <cns-device-info>
                <cns-device-name>c7200-2</cns-device-name>
                <cns-extended-attr
name="IOSconfigtemplate">http://sjc-filer01a-web/users-b/boniface/published/imgsvr_websvc/
sanity.test.cfgtpl/startup.7200-21.cfgtpl</cns-extended-attr>
                <cns-extended-attr name="IOShostname">c7200-2</cns-extended-attr>
                <cns-extended-attr name="IOSConfigID">c7200-2</cns-extended-attr>
                <cns-extended-attr name="IOSEventID">c7200-2</cns-extended-attr>
                <dev-image-information>
                    <image-id>c7200-2</image-id>

<activation-template>http://sjc-filer01a-web/users-b/boniface/published/imgsvr_websvc/sani
ty.test.cfgtpl/activation.7200-21.cfgtpl</activation-template>
```

```
                    <dev-image-info>
                        <image-name>7200-21</image-name>
                        <distribution overwrite="no" erase-flash="yes" activate="true">
                            <destination>slot0:c7200-tk8ea-mz.geo_20030721</destination>

<location>ftp://root:blender@cfg-earms1/images/c7200-tk8ea-mz.geo_20030721.T</location>
                        </distribution>
                    </dev-image-info>
                </dev-image-information>
            </cns-device-info>

      <cns-group-info>
        <cns-group-name>test</cns-group-name>
        <cns-group-member>c7200-2</cns-group-member>
      </cns-group-info>

        </NSM-DATA>
    </cns-element-data>
</cns-bulk-upload>
```

Distribute and Activate 7200-27

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-bulk-upload SYSTEM "BulkUpload.dtd">
<cns-bulk-upload stop-on-error="false">
    <cns-element-data>
        <NSM-DATA op-type="add" validate-data="true">
            <cns-device-info>
                <cns-device-name>c7200-2</cns-device-name>
                <cns-extended-attr
name="IOSconfigtemplate">http://sjc-filer01a-web/users-b/boniface/published/imgsvr_websvc/
sanity.test.cfgtpl/startup.7200-27.cfgtpl</cns-extended-attr>
                <cns-extended-attr name="IOShostname">c7200-2</cns-extended-attr>
                <cns-extended-attr name="IOSConfigID">c7200-2</cns-extended-attr>
                <cns-extended-attr name="IOSEventID">c7200-2</cns-extended-attr>
                <dev-image-information>
                    <image-id>c7200-2</image-id>

<activation-template>http://sjc-filer01a-web/users-b/boniface/published/imgsvr_websvc/sani
ty.test.cfgtpl/activation.7200-27.cfgtpl</activation-template>
                    <dev-image-info>
                        <image-name>7200-27</image-name>
                        <distribution overwrite="no" erase-flash="yes" activate="true">
                            <destination>slot0:c7200-tk8ea-mz.geo_20030727</destination>

<location>ftp://root:blender@cfg-earms1/images/c7200-tk8ea-mz.geo_20030727.T</location>
                        </distribution>
                    </dev-image-info>
                </dev-image-information>
            </cns-device-info>

      <cns-group-info>
        <cns-group-name>test</cns-group-name>
        <cns-group-member>c7200-2</cns-group-member>
      </cns-group-info>

        </NSM-DATA>
    </cns-element-data>
</cns-bulk-upload>
```

7200-21 Image

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-bulk-upload SYSTEM "BulkUpload.dtd">
```

```
<cns-bulk-upload stop-on-error="false">
<cns-element-data>
<IMAGE-DATA op-type="add">
  <image>
    <name>7200-21</name>
    <image-info image-type="IOS">
    <img-name>C7200-TK8EA-MZ</img-name>
    <img-chksum>4404252661ae36f605b2a0556887cdc3</img-chksum>
    <software-version>12.3(20030722:022836)</software-version>
    <system-description>Cisco Internetwork Operating System Software IOS (tm) 7200
Software (C7200-TK8EA-MZ), Experimental Version 12.3(20030722:022836)
[anrichar-georgia-20030721 101] Copyright (c) 1986-2003 by cisco Systems, Inc. Compiled
Mon 21-Jul-03 20:11 by anrichar</system-description>
    <file-byte-size>12425392</file-byte-size>
    <platform-family-name>C7200</platform-family-name>

<img-location>ftp://root:blender@cfg-earms1/images/c7200-tk8ea-mz.geo_20030721.T</img-loca
tion>
    </image-info>
  </image>
  <image>
    <name>7200-27</name>
    <image-info image-type="IOS">
    <img-name>C7200-TK8EA-MZ</img-name>
    <img-chksum>35bc79ea3d92fd330996d859a512f6c4</img-chksum>
    <software-version>12.3(20030728:055627)</software-version>
    <system-description>Cisco Internetwork Operating System Software IOS (tm) 7200
Software (C7200-TK8EA-MZ), Experimental Version 12.3(20030728:055627)
[anrichar-georgia-20030727 101] Copyright (c) 1986-2003 by cisco Systems, Inc. Compiled
Sun 27-Jul-03 23:37 by anrichar</system-description>
    <file-byte-size>12423420</file-byte-size>
    <platform-family-name>C7200</platform-family-name>

<img-location>ftp://root:blender@cfg-earms1/images/c7200-tk8ea-mz.geo_20030727.T</img-loca
tion>
    </image-info>
  </image>
</IMAGE-DATA>
</cns-element-data>
</cns-bulk-upload>
```

imgw-device

```
ces-docdev-1.cisco.com
Port: 1807

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cns-bulk-upload SYSTEM "BulkUpload.dtd">
<cns-bulk-upload stop-on-error="false">
<cns-element-data>
<NSM-DATA op-type="add" validate-data="false">
<cns-device-info dev-type="imgw">
  <cns-device-name>imgw-1</cns-device-name>
  <cns-extended-attr name="IOSconfigtemplate">XX.cfgtpl</cns-extended-attr>
  <cns-extended-attr name="IOSConfigID">imgw-1</cns-extended-attr>
  <cns-extended-attr name="IOSEventID">imgw-1</cns-extended-attr>
  <dev-image-information>
      <image-id>imgw-1</image-id>
  </dev-image-information>
  <imgw-data>
      <gateway-id>cns-stg6-1-l.cisco.com</gateway-id>
      <device-type>IOS</device-type>
      <simulation-agent>IMAGEAGENT</simulation-agent>
      <simulation-agent>CONFIGAGENT</simulation-agent>
```

```
        <hop-information>
            <hop-type>IOS_LOGIN</hop-type>
            <ip-address>10.92.1.193</ip-address>
            <port>0000</port>
            <username>admin</username>
            <password>localuser</password>
        </hop-information>
        <hop-information>
            <hop-type>IOS_EN</hop-type>
            <ip-address />
            <port />
            <username />
            <password />
        </hop-information>
    </imgw-data>
</cns-device-info>
</NSM-DATA>
</cns-element-data>
</cns-bulk-upload>
```

Image Web Service WSDL

The WSDL document defining the service methods and types is not downloadable over HTTP from the server for security reasons, but it can be downloaded from the Cisco Configuration Engine through SFTP by a root user from *<install base>/CSCOcnsie/WEB-INF/classes/imgsrv.wsdl*, where:

for Linux: **<install base> = /opt**

for Solaris: **<install base> = <as defined during installation>**

The following example is the API expressed in WSDL.

Note Object types referenced are self-explanatory by their names: viewing the WSDL document in a WSDL-browser will reveal the tree structure of nested types (i.e. **CnsMessage**).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:imgsrv"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns6="http://evaluation.job.websvc.imgsrv.cns.cisco.com"
xmlns:tns5="http://distribution.job.websvc.imgsrv.cns.cisco.com"
xmlns:tns4="http://job.websvc.imgsrv.cns.cisco.com"
xmlns:tns3="http://imgsrv.cns.cisco.com"
xmlns:tns2="http://msgobject.message.imgsrv.cns.cisco.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:intf="urn:imgsrv"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:impl="urn:imgsrv"
xmlns="http://schemas.xmlsoap.org/wsdl/">
 <wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:imgsrv">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_xsd_string">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
     </restriction>
    </complexContent>
   </complexType>
   <complexType name="ArrayOf_tns2_CnsMessage">
    <complexContent>
```

```
      <restriction base="soapenc:Array">
       <attribute ref="soapenc:arrayType" wsdl:arrayType="tns2:CnsMessage[]"/>
      </restriction>
     </complexContent>
    </complexType>
   </schema>
   <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://msgobject.message.imgsrv.cns.cisco.com">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="ErrorInfo">
     <sequence>
      <element name="additionalInfo" nillable="true" type="xsd:string"/>
      <element name="errorCode" nillable="true" type="xsd:string"/>
      <element name="errorMessage" nillable="true" type="xsd:string"/>
      <element name="messageID" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="ImageStatus">
     <sequence>
      <element name="errorInfo" nillable="true" type="tns2:ErrorInfo"/>
      <element name="imageName" nillable="true" type="xsd:string"/>
      <element name="status" nillable="true" type="xsd:string"/>
      <element name="version" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="ReloadNotify">
     <sequence>
      <element name="currentTime" nillable="true" type="xsd:string"/>
      <element name="eventType" nillable="true" type="xsd:string"/>
      <element name="imageName" nillable="true" type="xsd:string"/>
      <element name="reason" nillable="true" type="xsd:string"/>
      <element name="reloadTime" nillable="true" type="xsd:string"/>
      <element name="userName" nillable="true" type="xsd:string"/>
      <element name="version" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="Notify">
     <sequence>
      <element name="imageStatus" nillable="true" type="tns2:ImageStatus"/>
      <element name="reloadNotify" nillable="true" type="tns2:ReloadNotify"/>
     </sequence>
    </complexType>
    <complexType name="DeviceName">
     <sequence>
      <element name="hostName" nillable="true" type="xsd:string"/>
      <element name="imageID" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="CISFile">
     <sequence>
      <element name="fileType" nillable="true" type="xsd:string"/>
      <element name="fullName" nillable="true" type="xsd:string"/>
      <element name="modDate" nillable="true" type="xsd:string"/>
      <element name="name" nillable="true" type="xsd:string"/>
      <element name="owner" nillable="true" type="xsd:string"/>
      <element name="readFlag" nillable="true" type="xsd:string"/>
      <element name="runFrom" nillable="true" type="xsd:string"/>
      <element name="runSize" nillable="true" type="xsd:string"/>
      <element name="runnableImage" nillable="true" type="xsd:string"/>
      <element name="size" nillable="true" type="xsd:string"/>
      <element name="writeFlag" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="Directory">
```

```
    <sequence>
     <element name="directory" nillable="true" type="tns2:Directory"
maxOccurs="unbounded"/>
     <element name="file" nillable="true" type="tns2:CISFile" maxOccurs="unbounded"/>
     <element name="fullName" nillable="true" type="xsd:string"/>
     <element name="modDate" nillable="true" type="xsd:string"/>
     <element name="name" nillable="true" type="xsd:string"/>
     <element name="owner" nillable="true" type="xsd:string"/>
     <element name="readFlag" nillable="true" type="xsd:string"/>
     <element name="size" nillable="true" type="xsd:string"/>
     <element name="writeFlag" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="FileSystem">
    <sequence>
     <element name="directory" nillable="true" type="tns2:Directory"/>
     <element name="freespace" nillable="true" type="xsd:string"/>
     <element name="name" nillable="true" type="xsd:string"/>
     <element name="readable" nillable="true" type="xsd:string"/>
     <element name="size" nillable="true" type="xsd:string"/>
     <element name="type" nillable="true" type="xsd:string"/>
     <element name="writeable" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="FileSystemList">
    <sequence>
     <element name="fileSystems" nillable="true" type="tns2:FileSystem"
maxOccurs="unbounded"/>
    </sequence>
   </complexType>
   <complexType name="HardwareInfo">
    <sequence>
     <element name="hwRevision" nillable="true" type="xsd:string"/>
     <element name="hwRework" nillable="true" type="xsd:string"/>
     <element name="hwSerial" nillable="true" type="xsd:string"/>
     <element name="ioMemSize" nillable="true" type="xsd:string"/>
     <element name="mainMemSize" nillable="true" type="xsd:string"/>
     <element name="midplaneVersion" nillable="true" type="xsd:string"/>
     <element name="platformName" nillable="true" type="xsd:string"/>
     <element name="processorRev" nillable="true" type="xsd:string"/>
     <element name="processorType" nillable="true" type="xsd:string"/>
     <element name="vendor" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="RunningImageInfo">
    <sequence>
     <element name="bootVariable" nillable="true" type="xsd:string"/>
     <element name="bootldrVariable" nillable="true" type="xsd:string"/>
     <element name="configReg" nillable="true" type="xsd:string"/>
     <element name="configRegNextBoot" nillable="true" type="xsd:string"/>
     <element name="configVariable" nillable="true" type="xsd:string"/>
     <element name="imageFile" nillable="true" type="xsd:string"/>
     <element name="imageMD5" nillable="true" type="xsd:string"/>
     <element name="returnToROMReason" nillable="true" type="xsd:string"/>
     <element name="returnToROMTime" nillable="true" type="xsd:string"/>
     <element name="startedAt" nillable="true" type="xsd:string"/>
     <element name="versionString" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="ImageInventoryReport">
    <sequence>
     <element name="currentSystemTime" nillable="true" type="xsd:string"/>
     <element name="deviceName" nillable="true" type="tns2:DeviceName"/>
     <element name="fileSystemList" nillable="true" type="tns2:FileSystemList"/>
```

```
      <element name="hardwareInfo" nillable="true" type="tns2:HardwareInfo"/>
      <element name="runningImageInfo" nillable="true" type="tns2:RunningImageInfo"/>
     </sequence>
    </complexType>
    <complexType name="ImageInventoryRsp">
     <sequence>
      <element name="errorInfo" nillable="true" type="tns2:ErrorInfo"/>
      <element name="imageInventoryReport" nillable="true"
type="tns2:ImageInventoryReport"/>
      <element name="status" nillable="true" type="xsd:string"/>
      <element name="version" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="Response">
     <sequence>
      <element name="imageInventoryRsp" nillable="true" type="tns2:ImageInventoryRsp"/>
     </sequence>
    </complexType>
    <complexType name="SenderCredentials">
     <sequence>
      <element name="passWord" nillable="true" type="xsd:string"/>
      <element name="userName" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="InitiatorCredentials">
     <sequence>
      <element name="passWord" nillable="true" type="xsd:string"/>
      <element name="userName" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="InitiatorInfo">
     <sequence>
      <element name="initiatorCredentials" nillable="true"
type="tns2:InitiatorCredentials"/>
      <element name="trigger" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="ImageSessionStart">
     <sequence>
      <element name="initiatorInfo" nillable="true" type="tns2:InitiatorInfo"/>
      <element name="version" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="SessionControl">
     <sequence>
      <element name="imageSessionStart" nillable="true" type="tns2:ImageSessionStart"/>
     </sequence>
    </complexType>
    <complexType name="CnsMessage">
     <sequence>
      <element name="messageID" nillable="true" type="xsd:string"/>
      <element name="msgBoot" nillable="true" type="xsd:anyType"/>
      <element name="msgImageInventoryResponse" nillable="true" type="xsd:anyType"/>
      <element name="msgImageSessionStart" nillable="true" type="xsd:anyType"/>
      <element name="msgImageStatus" nillable="true" type="xsd:anyType"/>
      <element name="msgReloadNotify" nillable="true" type="xsd:anyType"/>
      <element name="msgType" nillable="true" type="xsd:string"/>
      <element name="notify" nillable="true" type="tns2:Notify"/>
      <element name="response" nillable="true" type="tns2:Response"/>
      <element name="senderCredentials" nillable="true" type="tns2:SenderCredentials"/>
      <element name="sessionControl" nillable="true" type="tns2:SessionControl"/>
      <element name="version" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
```

```
  </schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://imgsrv.cns.cisco.com">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="CISException">
    <sequence/>
   </complexType>
  </schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://job.websvc.imgsrv.cns.cisco.com">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="JobDTO">
    <sequence>
     <element name="activation" type="xsd:boolean"/>
     <element name="batchSize" type="xsd:int"/>
     <element name="date" nillable="true" type="xsd:dateTime"/>
     <element name="description" nillable="true" type="xsd:string"/>
     <element name="distribution" type="xsd:boolean"/>
     <element name="evalutaion" type="xsd:boolean"/>
     <element name="groupNames" nillable="true" type="xsd:string" maxOccurs="unbounded"/>
     <element name="imageIds" nillable="true" type="xsd:string" maxOccurs="unbounded"/>
    </sequence>
   </complexType>
   <complexType name="ImageIdJobId">
    <sequence>
     <element name="imageId" nillable="true" type="xsd:string"/>
     <element name="jobId" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="DistributedImagesEntityDTO">
    <sequence>
     <element name="activatedImageName" nillable="true" type="xsd:string"/>
     <element name="deviceName" nillable="true" type="xsd:string"/>
     <element name="distributedImageNames" nillable="true" type="xsd:string"
maxOccurs="unbounded"/>
    </sequence>
   </complexType>
   <complexType name="SubmitStatus">
    <sequence>
     <element name="errorInfo" nillable="true" type="tns2:ErrorInfo"/>
     <element name="inProgress" nillable="true" type="tns4:ImageIdJobId"
maxOccurs="unbounded"/>
     <element name="jobId" nillable="true" type="xsd:string"/>
     <element name="submissions" nillable="true" type="tns4:DistributedImagesEntityDTO"
maxOccurs="unbounded"/>
    </sequence>
   </complexType>
   <complexType name="JobStatusDTO">
    <sequence>
     <element name="description" nillable="true" type="xsd:string"/>
     <element name="deviceStatus" nillable="true" type="tns5:DeviceStatusDTO"
maxOccurs="unbounded"/>
     <element name="id" nillable="true" type="xsd:string"/>
     <element name="startTime" nillable="true" type="xsd:dateTime"/>
     <element name="status" nillable="true" type="xsd:string"/>
     <element name="type" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://evaluation.job.websvc.imgsrv.cns.cisco.com">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ActivationEvaluationDTO">
    <sequence>
```

```
      <element name="destination" nillable="true" type="xsd:string"/>
      <element name="errorInfo" nillable="true" type="xsd:string"/>
      <element name="imageName" nillable="true" type="xsd:string"/>
      <element name="reason" nillable="true" type="xsd:string"/>
      <element name="required" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="DistributionEvaluationDTO">
     <sequence>
      <element name="destination" nillable="true" type="xsd:string"/>
      <element name="errorInfo" nillable="true" type="xsd:string"/>
      <element name="hasEnoughSpace" nillable="true" type="xsd:string"/>
      <element name="imageName" nillable="true" type="xsd:string"/>
      <element name="reason" nillable="true" type="xsd:string"/>
      <element name="remainingSpace" nillable="true" type="xsd:long"/>
      <element name="required" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="DeviceEvaluationDTO">
     <sequence>
      <element name="activationEvals" nillable="true" type="tns6:ActivationEvaluationDTO"
maxOccurs="unbounded"/>
      <element name="distributionEvals" nillable="true"
type="tns6:DistributionEvaluationDTO" maxOccurs="unbounded"/>
      <element name="imageId" nillable="true" type="xsd:string"/>
      <element name="reachable" type="xsd:boolean"/>
     </sequence>
    </complexType>
   </schema>
   <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://distribution.job.websvc.imgsrv.cns.cisco.com">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="DeviceStatusDTO">
     <sequence>
      <element name="deviceEvaluation" nillable="true" type="tns6:DeviceEvaluationDTO"/>
      <element name="deviceName" nillable="true" type="xsd:string"/>
      <element name="imageStatus" nillable="true" type="tns2:ImageStatus"
maxOccurs="unbounded"/>
      <element name="reloadNotify" nillable="true" type="tns2:ReloadNotify"/>
     </sequence>
    </complexType>
   </schema>
  </wsdl:types>

    <wsdl:message name="deleteDeviceResponse">

    </wsdl:message>

    <wsdl:message name="restartJobRequest">

       <wsdl:part name="in0" type="xsd:string"/>

    </wsdl:message>

    <wsdl:message name="cancelJobResponse">

    </wsdl:message>

    <wsdl:message name="evaluateJobResponse">

       <wsdl:part name="evaluateJobReturn" type="tns4:SubmitStatus"/>

    </wsdl:message>
```

```
<wsdl:message name="submitJobRequest">
   <wsdl:part name="in0" type="tns4:JobDTO"/>
</wsdl:message>
<wsdl:message name="evaluateJobRequest">
   <wsdl:part name="in0" type="tns4:JobDTO"/>
</wsdl:message>
<wsdl:message name="restartJobResponse">
</wsdl:message>
<wsdl:message name="deleteImageResponse">
</wsdl:message>
<wsdl:message name="listJobIdsRequest">
</wsdl:message>
<wsdl:message name="bulkUploadResponse">
   <wsdl:part name="bulkUploadReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="listJobIdsResponse">
   <wsdl:part name="listJobIdsReturn" type="intf:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="deleteGroupRequest">
   <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="bulkUploadRequest">
   <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="CISException">
   <wsdl:part name="fault" type="tns3:CISException"/>
</wsdl:message>
<wsdl:message name="submitJobResponse">
   <wsdl:part name="submitJobReturn" type="tns4:SubmitStatus"/>
</wsdl:message>
<wsdl:message name="getImageInventoryReportsRequest">
   <wsdl:part name="in0" type="intf:ArrayOf_xsd_string"/>
```

```
   </wsdl:message>

   <wsdl:message name="getJobDetailStatusRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="getJobDetailStatusResponse">

      <wsdl:part name="getJobDetailStatusReturn" type="tns4:JobStatusDTO"/>

   </wsdl:message>

   <wsdl:message name="getImageInventoryReportsResponse">

      <wsdl:part name="getImageInventoryReportsReturn"
type="intf:ArrayOf_tns2_CnsMessage"/>

   </wsdl:message>

   <wsdl:message name="deleteDeviceRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="cancelJobRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="getJobStatusRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="stopJobRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="deleteGroupResponse">

   </wsdl:message>

   <wsdl:message name="getJobStatusResponse">

      <wsdl:part name="getJobStatusReturn" type="xsd:string"/>

    </wsdl:message>

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="deleteImageRequest">

      <wsdl:part name="in0" type="xsd:string"/>

   </wsdl:message>
```

```
   <wsdl:message name="stopJobResponse">

   </wsdl:message>

   <wsdl:portType name="ImageWebService">

      <wsdl:operation name="getImageInventoryReports" parameterOrder="in0">

         <wsdl:input name="getImageInventoryReportsRequest"
message="intf:getImageInventoryReportsRequest"/>

         <wsdl:output name="getImageInventoryReportsResponse"
message="intf:getImageInventoryReportsResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="submitJob" parameterOrder="in0">

         <wsdl:input name="submitJobRequest" message="intf:submitJobRequest"/>

         <wsdl:output name="submitJobResponse" message="intf:submitJobResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="evaluateJob" parameterOrder="in0">

         <wsdl:input name="evaluateJobRequest" message="intf:evaluateJobRequest"/>

         <wsdl:output name="evaluateJobResponse" message="intf:evaluateJobResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="listJobIds">

         <wsdl:input name="listJobIdsRequest" message="intf:listJobIdsRequest"/>

         <wsdl:output name="listJobIdsResponse" message="intf:listJobIdsResponse"/>

      </wsdl:operation>

      <wsdl:operation name="stopJob" parameterOrder="in0">

         <wsdl:input name="stopJobRequest" message="intf:stopJobRequest"/>

         <wsdl:output name="stopJobResponse" message="intf:stopJobResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="restartJob" parameterOrder="in0">

         <wsdl:input name="restartJobRequest" message="intf:restartJobRequest"/>

         <wsdl:output name="restartJobResponse" message="intf:restartJobResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>
```

```
      </wsdl:operation>

      <wsdl:operation name="cancelJob" parameterOrder="in0">

         <wsdl:input name="cancelJobRequest" message="intf:cancelJobRequest"/>

         <wsdl:output name="cancelJobResponse" message="intf:cancelJobResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="getJobStatus" parameterOrder="in0">

         <wsdl:input name="getJobStatusRequest" message="intf:getJobStatusRequest"/>

         <wsdl:output name="getJobStatusResponse" message="intf:getJobStatusResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="getJobDetailStatus" parameterOrder="in0">

         <wsdl:input name="getJobDetailStatusRequest"
message="intf:getJobDetailStatusRequest"/>

         <wsdl:output name="getJobDetailStatusResponse"
message="intf:getJobDetailStatusResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="deleteDevice" parameterOrder="in0">

         <wsdl:input name="deleteDeviceRequest" message="intf:deleteDeviceRequest"/>

         <wsdl:output name="deleteDeviceResponse" message="intf:deleteDeviceResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="deleteImage" parameterOrder="in0">

         <wsdl:input name="deleteImageRequest" message="intf:deleteImageRequest"/>

         <wsdl:output name="deleteImageResponse" message="intf:deleteImageResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

      <wsdl:operation name="deleteGroup" parameterOrder="in0">

         <wsdl:input name="deleteGroupRequest" message="intf:deleteGroupRequest"/>

         <wsdl:output name="deleteGroupResponse" message="intf:deleteGroupResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>
```

```
      </wsdl:operation>

      <wsdl:operation name="bulkUpload" parameterOrder="in0">

         <wsdl:input name="bulkUploadRequest" message="intf:bulkUploadRequest"/>

         <wsdl:output name="bulkUploadResponse" message="intf:bulkUploadResponse"/>

         <wsdl:fault name="CISException" message="intf:CISException"/>

      </wsdl:operation>

   </wsdl:portType>

   <wsdl:binding name="imgsrvSoapBinding" type="intf:ImageWebService">

      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="getImageInventoryReports">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="getImageInventoryReportsRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="getImageInventoryReportsResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="submitJob">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="submitJobRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="submitJobResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>
```

```
         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="evaluateJob">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="evaluateJobRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="evaluateJobResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="listJobIds">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="listJobIdsRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="listJobIdsResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

      </wsdl:operation>

      <wsdl:operation name="stopJob">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="stopJobRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>
```

```
         </wsdl:input>

         <wsdl:output name="stopJobResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="restartJob">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="restartJobRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="restartJobResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="cancelJob">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="cancelJobRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="cancelJobResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>
```

```
         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="getJobStatus">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="getJobStatusRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="getJobStatusResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="getJobDetailStatus">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="getJobDetailStatusRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="getJobDetailStatusResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="deleteDevice">
```

```
         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="deleteDeviceRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="deleteDeviceResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

         <wsdlsoap:operation soapAction=""/>

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="deleteImage">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="deleteImageRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="deleteImageResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>
```

```
         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="deleteGroup">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="deleteGroupRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="deleteGroupResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>

      <wsdl:operation name="bulkUpload">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="bulkUploadRequest">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:input>

         <wsdl:output name="bulkUploadResponse">

            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:output>

         <wsdl:fault name="CISException">

            <wsdlsoap:fault use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:imgsrv"/>

         </wsdl:fault>

      </wsdl:operation>
```

```
   </wsdl:binding>

   <wsdl:service name="ImageWebServiceService">

      <wsdl:port name="imgsrv" binding="intf:imgsrvSoapBinding">

         <wsdlsoap:address location="http://localhost/cns/services/imgsrv"/>

      </wsdl:port>

   </wsdl:service>

</wsdl:definitions>
```

Spreadsheet Bulk Upload

You can use **imgsrv** Web Service to do a bulk creation of devices with Microsoft Excel.

For information about how to use this feature, see the README file that is located in *<install_path>/CE-SDK/tools/excel/README*.

CHAPTER 6

Namespace Administration, Group Administration, and Notification APIs

This section introduces the following three groups of APIs:

- Namespace administration API for managing Namespace.
- Group administration API for supporting hierarchical grouping.
- Notification API for tracking dynamic group changes and namespace changes.

These APIs are also referred to as remote APIs because they communicate with the servlets on the Cisco Configuration Engine using HTTP.

These sections present the Java version of each API.

The corresponding C++ version of each API is shown in the associated reference sections beginning with "Namespace Administration API Reference" section on page 6-51.

Namespace Administration API

The Namespace Administration interface allows users to create their own application namespace and define subject mappings for the namespace.

Class NamespaceAdminFactory

The factory NamespaceAdminFactory creates a NamespaceAdmin object that meets the criteria specified by a Properties object. The Properties object contains a set of name-value pairs. An example of name-value pair is provider-remote. The Name-Value property pairs supported for the NamespaceAdminFactory in Cisco Configuration Engine are listed in "Properties Supported" section on page 6-56.

Constructor

It has two constructors; the first takes a Properties object as input and the second a property filename as input.

Method

NamespaceAdminFactory(Properties myProp)

Input parameter

myProp – Properties object that specifies the name-value pairs for creating the Namespace administration object.

Method

NamespaceAdminFactory(String myPropFile)

Input parameter

myPropFile – Name of the property file.

Create

This method creates a NamespaceAdmin object with the specified properties. The Namespace administration object provides the namespace administration functions.

Method

NamespaceAdmin create()

Input

None.

Return

NamespaceAdmin object.

Delete

This method destroys a NamespaceAdmin object and returns unused system resource.

Method

void delete(NamespaceAdmin admin)

Input Parameter

admin – the NamespaceAdmin object to be destroyed.

Class NamespaceAdmin

Add Namespace

The add-Namespace operation creates a namespace object, when given the identifier of the namespace.

Method void addNamespace (String namespaceId)

Input parameter

namespaceId This is a String that uniquely identifies a namespace to be created.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **NamespaceAlreadyExistsException** – Namespace already existed.
- **InvalidInputException** – Any invalid input, such as NULL and “”.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Namespace Deletion

The delete-Namespace operation deletes a namespace object.

Note This is true only when a namespace does not contain subject mappings or group hierarchy.

Method void delNamespace (String namespaceId)

Input parameter

namespaceId This is a String that uniquely identifies a namespace to be deleted.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – Any invalid inputs, such as NULL and “”.
- **NamespaceNotFoundException** – Namespace not found.
- **GroupExistsException** – Namespace contains a group hierarchy.
- **SubjectAlreadyExistsException** – Subject mapping found.
- **GroupFoundException** – Namespace contains a group hierarchy.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Clone Namespace

This method is responsible for making a copy of the given Namespace, but not including the groups.

Note This method clones only the subjects, not the group hierarchy associated with the namespace.

Method void cloneNamespace (String namespaceId, String newNamespaceId)

Input parameter

namespaceId This is a String that uniquely identifies a namespace to be copied.

newNamespaceId This is the name of the new namespace.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – Any invalid inputs, such as NULL, “”, or **newNamesapceId** and **namespaceId** are the same.
- **NamespaceNotFoundException** – Original namespace not found.
- **NamespaceAlreadyExistsException** – New Namespace already exists.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

List All Namespaces

The list-Namespace operation returns an array of existing namespace Ids.

Method String[ ] listNamespaces ()

Input parameter None.

Return An array of namespace IDs in Strings.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **MessageFormatException** – Problem with message XML message syntax to be sent between the server and the remote API.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

List Selected Namespaces

The listNamespace operation returns an array of existing namespace Ids which meet certain directory search criteria denoted by the filter parameter.

Method	String[ ] listNamespaces (String filter)
Input parameter	
filter	A string input that denotes a directory search filter for the listNamespaces operation. A typical input of the form "ou=N1*" will returns all namespaces with name starting with the string N1.
Return	An array of namespace IDs in Strings that meet the criteria denoted by filter.
Exceptions	This method could throw one of the following exceptions: • **DatastoreAccessException** – General data-store access problem that is not described below. • **MessageFormatException** – Problem with message XML message syntax to be sent between the server and the remote API. Additional exceptions that could be thrown by the remote API: **CommunicationException** – General server access problem.

Add Subject in Namespace

This operation adds a single subject to a given namespace.

Method	void addSubject (String namespaceId, String subject)
Input parameter	
namespaceId	This is a string that uniquely identifies a namespace.
subject	A string subject to be added.
Return	None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – see Table 6-2 for basic rules.
- **NamespaceNotFoundException** – Namespace not found.
- **SubjectAlreadyExistsException** – Subject already exists.
- **MessageFormatException** – Problem with message syntax.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

Delete Subject in Namespace

This operation deletes a single subject from a given namespace.

Method void delSubject (String namespaceId, String subject)

Input parameter

namespaceId This is a string that uniquely identifies a namespace.

subject A string subject to be deleted.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – see Table 6-2 for basic rules.
- **NamespaceNotFoundException** – Namespace not found.
- **SubjectNotFoundException** – Subject not found.
- **MessageFormatException** – Problem with message syntax.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

List All Subjects in Namespace

This operation returns all the subjects of a given namespace.

Method	String[ ] listSubjects (String namespaceId)
Input parameter	
namespaceId	This is a string that uniquely identifies a namespace.
Return	An array of Strings that lists the subjects.
Exceptions	This method could throw one of the following exceptions: • **DatastoreAccessException** – General data-store access problem that is not described below. • **InvalidInputException** – see Table 6-2 for basic rules. • **NamespaceNotFoundException** – Namespace not found • **MessageFormatException** – Problem with XML message syntax to be sent between the server and the remote API. Additional exceptions that could be thrown by the remote API: **CommunicationException** – General server access problem.

List Selected Subjects in a Namespace

This operation returns selected subjects of a given namespace. The subjects should meet the search criteria denoted by filter.

Method	String[ ] listSubjects (String namespaceId, String filter)
Input parameter	
namespaceId	This is a string that uniquely identifies a namespace.
filter	A string input which denotes a directory search filter for the listSubjects operation. A typical input of the form “cn=S1*” will returns all subjects with name starting with the string S1 under given namespace.
Return	An array of Strings that lists the subjects.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **InvalidInputException** – see Table 6-2 for basic rules.
- **NamespaceNotFoundException** – Namespace not found.
- **MessageFormatException** – Problem with XML message syntax to be sent between the server and the remote API.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

Add Mapping for Given Subject Into Namespace

Each subject has two mappings, one for publish and one for subscribe. Operation addSubjectMapping adds a publish- or subscribe-mapping for the given subject into the given namespace.

Method void addSubjectMapping (String namespaceId, int mapType, String subject, String[ ] subjectMapping, int resolveMode)

Input parameter

namespaceId This is a string that uniquely identifies a namespace.

mapType Indicate the type of subject mapping to be added, PUBLISH or SUBSCRIBE.

subject This parameter identifies the associated subject for the mapping be added; for example, *cisco.mgmt.cns.config.load*.

subjectMapping This is a String array that stores the mapping to be added for the given subject.

resolveMode An integer constant which defines the NSM resolve mode, ALGORITHMIC or NON_ALGORITHMIC, of the mapping to be added.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException**
  - see Table 6-2 for basic rules.
  - NULL subjectMapping
  - mapType other than PUBLISH or SUBSCRIBE.
  - resolveMode other than ALGORITHMIC or NON_ALGORITHMIC.
- **NamespaceNotFoundException** – Namespace not found
- SubjectNotFoundException – Subject not defined yet in the given namespace.
- **MappingAlreadyExistsException** – Mapping already existed.
- **InvalidMappingException** – NULL or empty string "" in subjectMapping array entry.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Deleting Mapping from Namespace

Operation delSubjectMapping deletes a selected mapping from the given subject and namespace.

Method void delSubjectMapping (String namespaceId, int mapType, String subject)

Input parameter

namespaceId This is a string that uniquely identifies a namespace.

mapType Indicate the type of subject mapping to be deleted, PUBLISH, SUBSCRIBE, or BOTH.

subject This parameter is a string that identifies the subject for which the selected mapping is to be removed.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException**
  - see Table 6-2 for basic rules.
  - NULL subjectMapping
  - mapType other than PUBLISH or SUBSCRIBE.
- **NamespaceNotFoundException** – Namespace not found
- **SubjectNotFoundException** – Subject not found in the given namespace
- **MappingNotFoundException** – Mapping not found; for example, for a given mapType, subject, and namespace, there is no associated mapping found in the data-store.
- **MessageFormatException** – Problem with message syntax.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

List Subject Mapping

This operation returns the subject map of a given subject and mapping type under a namespace.

Method String[ ] listSubjectMappings (String namespaceId, int mapType, String subject)

Input parameter

namespaceId This is a string that uniquely identifies a namespace.

mapType Indicate the type of subject mapping to be retrieved: PUBLISH or SUBSCRIBE.

subject This is subject ID for which the mapping is returned.

Return An array of Strings that lists the mapping.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **InvalidInputException**
  - see Table 6-2 for basic rules.
  - mapType other than PUBLISH or SUBSCRIBE.
- **NamespaceNotFoundException** – Namespace not found
- **SubjectNotFoundException** – Subject not found in the given namespace
- **MappinpNotFoundException**; for example, for a given mapType, subject, and namespace, there is no associated mapping found in the data-store.
- **MessageFormatException** – Problem with message syntax.

Additional exceptions that could be thrown by the remote API:

CommunicationException – General server access problem.

Get Mapping Resolution Mode

This operation returns the resolve mode when given a specific subject, map type, and namespace.

Method int getResolveMode(String namespaceId, int mapType, String subject)

Input parameter

namespaceId This is a string that uniquely identifies a namespace.

mapType Indicate the type of subject mapping to be retrieved: PUBLISH or SUBSCRIBE.

subject This is subject ID for which the mapping is returned.

Return An integer that denotes either ALGORITHMIC or NON_ALGORITHMIC mode.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **InvalidInputException**
  - see Table 6-2 for basic rules.
  - mapType other than PUBLISH or SUBSCRIBE.
- **NamespaceNotFoundException** – Namespace not found
- **SubjectNotFoundException** – Subject not found in the given namespace
- **MappingNotFoundException** – Mapping not found for the given mapType
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Set Notification

This method provides the client application the capability of stopping and re-starting subsequent namespace administration operations from generating change notification events. In cases where the client application is expected to perform a great number of groups and devices re-arrangement operations, the client application could choose to stop notification temporarily (using setNotification(FALSE)), carry out the operations, and then re-enable notification (using setNotification(TRUE)) after. This could reduce the event traffic and avoid handling of a large number of notification events.

When re-enabling notification, this method sends a **global-change** notification to all subscribing applications, asking them to re-load all its subject mappings to the ones defined by the latest grouping and namespace definitions.

Method void setNotification (Boolean state)

Input parameter

state A boolean indicates whether to turn on or turn off the sending of notification event for the subsequent change operation.

Return None.

Exceptions Exception that can be thrown:

- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **GroupAdminException** to indicate general failure.
- **MessageFormatException** – Problem with message syntax.
- **CommunicationException** – General server access problem.
- **MessageFormatException** – Problem with message syntax.

Exception that can be thrown by the remote API:

CommunicationException – General server access problem.

Group Administration API

The Group Administration interface provides the programatic interface for applications that need to group devices. In order to support hierarchical grouping, the LDAP schema has been updated to define a **cnsGroup** based upon **OrganizationUnit**. See "Schema Changes" section on page 6-41 for details.

Grouping rules

Figure 6-1 Example of Hierarchical Groups in Namespace NS1

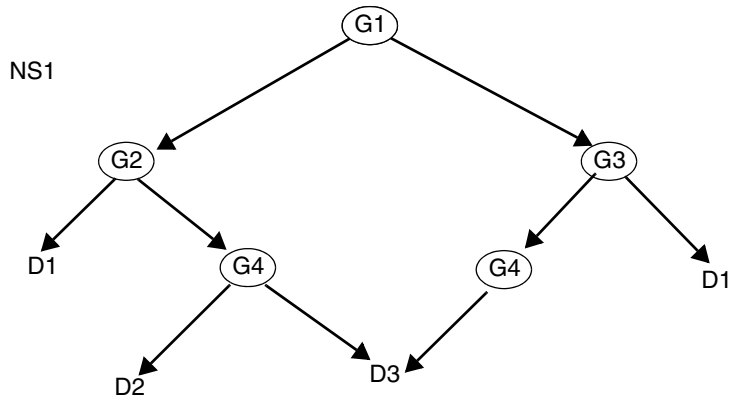


Figure 6-1 illustrates an example of hierarchical groups that are supported in Cisco Configuration Engine. There are a total of five groups and three devices. Note the following rules:

- Each namespace (or application) defines its own group hierarchies. See "NSM Support for Hierarchical Groups" section on page 6-41 for details.
- Device IDs must be unique across the Integration Bus; two different devices cannot have the same ID. However, the same device can be member of multiple groups, as in the case of device D1 being a member of groups G2 and G3.

- Group ID should be defined using an alpha-numeric string, and two group IDs cannot be the same if they have the same direct parent. As a result, the hierarchy could be made up of groups that have the same group ID; for example, G4.
- An absolute group ID is required in order to uniquely address a group. For example, the absolute group ID for the G4 on the left hand side is /NS1/G1/G2/G4 and the G4 on the right hand side /NS1/G1/G3/G4. The typical directory-reference notation used for file-system access is adopted here. Note that all absolute group IDs are qualified using the namespace ID. As a result, the group administration operation to be presented next can be used across namespace.

Note

For ease of illustration, the remaining examples illustrating group administration APIs will adopt the following conventions:

- All Group administration operation in the examples are assumed to be done in the default "config" namespace. As a result, all absolute group Id in the example will be prepended with the config namespace specification "/config".
- We use Id String that starts with a letter G to indicate group name and D to indicate device name. For example, G1, G2, and etc. are group names while D1, D2, and etc. are device names.

Class GroupAdminFactory

Similar to the NamespaceAdminFactory, the factory GroupAdminFactory creates the GroupAdmin object that meets the criteria specified by a Properties object. The Name-Value property pairs supported for the GroupAdminFactory is listed in "Properties Supported" section on page 6-61.

Constructor

It has two constructors; the first takes a Properties object as input and the second a property file name as input.

Method

GroupAdminFactory(Properties myProp)

Input Parameter

myProp – Properties object which specifies the name-value pairs for creating the Group administration object.

Method

GroupAdminFactory(String myPropFile)

Input Parameter

myPropFile – name of the property file.

create

This method creates a GroupAdmin object with the specified properties. The Group administration object provides the group administration functions.

Method

GroupAdmin create()

Input

None.

Return

GroupAdmin object

Delete

This method destroys a GroupAdmin object and returns unused system resource.

Method

void delete(GroupAdmin admin)

Input Parameter

admin – GroupAdmin object to be destroyed.

GroupMember

Inputs and outputs of the group administration API involve two main types, String and GroupMember. GroupMember is the base class of the two derived classes Element and Group, so a GroupMember array (GroupMember[]) can contain both Element objects and Group objects.

When managing device using the Group Administration API, user should use the element constructor Element("<deviceId>") to create the device object to be managed. Likewise, user should use the Group constructor Group("<groupId>") to create the group object.

The base class GroupMember provides the following public functions that can be used to access the ELEMENT/GROUP Id and absolute ELEMENT/GROUP Id in the returned GroupMember array of the listMember API.

public String getId()

public String getAbsoluteId()

Class GroupAdmin – Group Administration API

Add Members

Method **addMembers** provides the function for adding device or group members into the given group specified by **absGroupId** (absolute group ID).

Note Device must first be created using the device administration API before it can be added into a group.

Method	void addMembers (String absGroupId, GroupMember[] members)
Input parameter	
absGroupId	Parameter absGroupId is an absolute group ID that specifies the group to be modified. The elements in the absolute path must already exist in the directory for the operation to succeed.
members	This is the array of members to be created under absGroupId.
Return	None.

Exceptions

This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – See Table 6-2 for basic rules.
- **ParentNotFoundException** – Cannot locate the group absGroupId in the hierarchy.
- **InvalidMemberException** – if one of the following is true:
  - a device member has not been created in the data-store with the device management API.
  - When adding a group, resulting absolute group ID cannot violates the absolute group ID string-length limit.
  - See Table 6-2 for additional details.
- **MemberAlreadyExistsException** – Member already existed in group **absGroupId**.
- **MessageFormatException** – Problem with message syntax.

Additional exception that can be thrown when using the remote API:

CommunicationException – Server connection problem

Example one:

Create two initial groups G1 and G2 and add device D1 into G1 as follows:

addMembers("/config",[G1, G2]);

addMembers("/config/G1", [D1]);

Note the following use of notations to simplify the illustration:

- The absolute group Id "/config" indicates that the groups are created at the root level of default namespace "config."
- [G1, G2] denotes an array of GroupMember, the first is the Group object G4 and the second the Group object G2.
- [D1] denotes an GroupMember array of one Element member D1.

Example two:

Throw InvalidInputException when adding a device without a valid absolute group ID as follows:

addMembers("/config",[D1]);

addMembers("G1",[D1]);

addMembers("/config/G1/",[D1]);

Example three:

Throw ParentNotFoundException when adding a device to a group that does not exist; for example, assuming there is no group "/config/G3" in the hierarchy:

addMembers("/config/G3",[D1]);

Delete Members

This is a recursive delete operation that deletes a list of members from within a given group (see Figure 6-2).

Method	void deleteMembers (String absGroupId, GroupMember[] members)
Input parameter	
absGroupId	This is an absolute group ID that identifies the group to be modified.
members	This is an array of **GroupMember** objects which, respectively, specify the sub-groups and devices inside absGroupId to be removed.
Return	None.

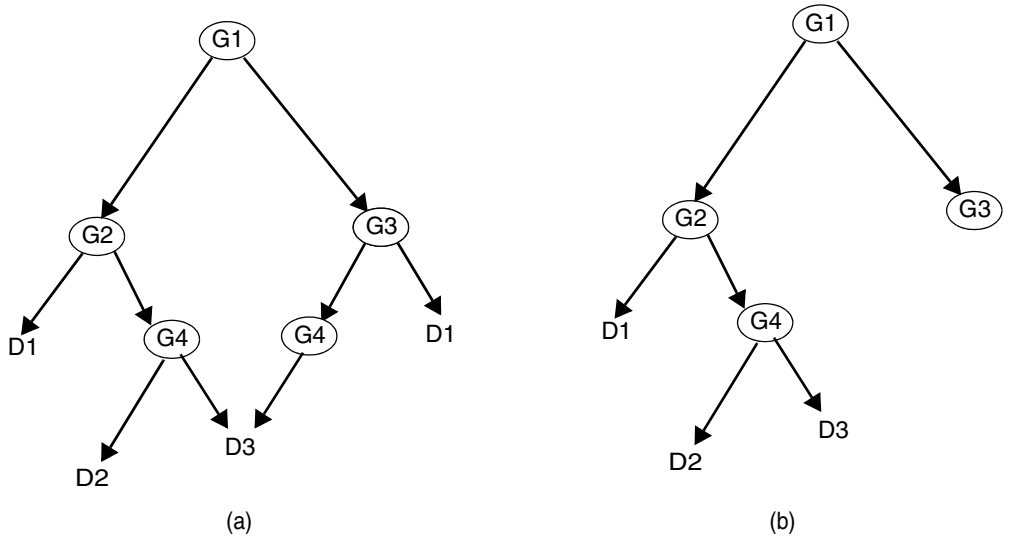
Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – See Table 6-2 for basic rules.
- **InvalidMemberException** – See Table 6-2 for basic rules.
- **ParentNotFoundException** – Cannot locate group specified by **absGroupId**.
- **MemberNotFoundException** – Member not found in group **absGroupId**.
- **MessageFormatException** – Problem with message syntax.

Additional exception that can be thrown by remote API:

CommunicationException – General server access problem.

Figure 6-2 Example of Delete Members



Example:

Given Figure 6-2 (a) as an example, the delete operation deleteMembers("/config/G1/G3", [G4,D1]) removes group G4 and device D1 from group /config/G1/G3, as shown in Figure 6-2 (b). The removal of group G4 is a recursive operation because member device D3 of group G4 is also removed.

Delete All Members

This is a recursive delete operation that removes all members from the given group.

Method void deleteAllMembers (String absGroupId)

Input parameter

absGroupId	This is an absolute group ID that identifies the group to be modified.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – General data-store access problem that is not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException** – See Table 6-2 for basic rules.
- **ParentNotFoundException** – Cannot locate group specified by absGroupId.
- **MessageFormatException** – Problem with message syntax.

Additional exception that can be thrown when using remote API:

CommunicationException – General server access problem

Move Members

This operation moves the given list of members from the source group into the destination group (see Figure 6-3). If there are sub-groups in the member list, the entire sub-group hierarchy is moved.

Method void moveMembers (String srcAbsGroupId, String destAbsGroupId, GroupMember[] members)

Input parameter

srcAbsGroupId	This is the absolute group ID that identifies the source group.
destAbsGroupId	This is the absolute group ID that identifies the destination group.
members	This is an array of GroupMember that identifies the sub-groups and devices to be moved from **srcAbsGroupId** into **destAbsGroupId**.

Return None.

Exceptions

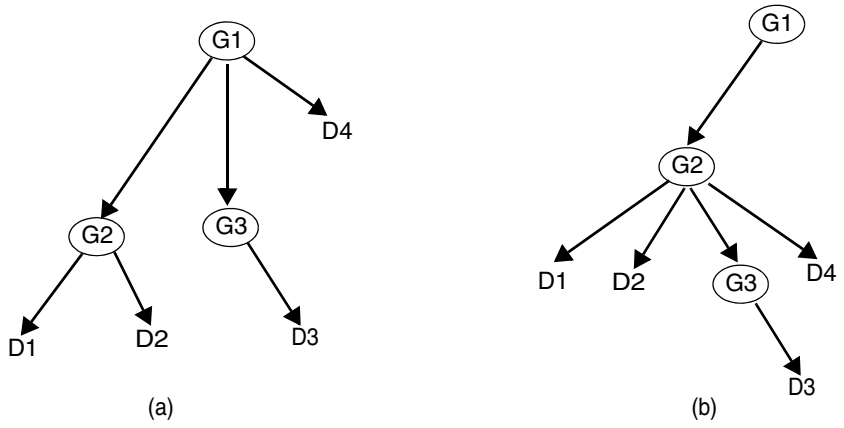
This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException**
  - See Table 6-2 for basic rules.
  - **srcAbsGroupId** and **destAbsGroupId** cannot be the same.
- **InvalidOperationException**
  - The group members in the **srcAbsGroupId** to be moved cannot be a parent or ancestor group of **destAbsGroup** because it will break the group hierarchical structure. See Example three given below.
  - In additional, one cannot move the destination group itself as in moveMembers("/config/G1","/config/G1/G2", [G2]);
- **InvalidMemberException**
  - See Table 6-2 for basic rules.
  - The sub-group member and its descendent groups to be moved cannot result into an absolute group ID that violates the absolute-group-ID string-length restriction.
- **ParentNotFoundException** – Cannot locate group specified by **srcAbsGroupId** or **destAbsGroupId**.
- MemberNotFoundException – Member not found in group **srcAbsGroupId**.
- **MemberAlreadyExistsException** – Sub-group member or device member already existed in **destAbsGroupId**.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Figure 6-3 Example of Move Members



Example one:

Figure 6-3 shows the operation of moving group G3 and device D4 into group G2: moveMembers("/config/G1", "/config/G1/G2", [G3, D4]). Figure 6-3 (a) and (b) shows the grouping structures before and after the move respectively.

Example two:

Using Figure 6-3 (b), the following two move operations respectively throw MemberNotFoundException and InvalidInputException:

moveMembers("/config/G1","/config/G1/G2",[D4])

moveMembers("/config/G1","/config/G1",[G2])

Example three:

The group members in srcAbsGroupId to be moved cannot be a **parent** or **ancestor** group of destAbsGroupId because it will break the group hierarchical structure. Using Figure 6-3 (b) as an example, it is not a valid operation to move group G2 into group G3; however, it is a valid operation to move device D3 into group G1 or G2.

List Members

This operation returns the selected (immediate or descendant) members of the given group.

Method	GroupMember[] listMembers (String absGroupId, String filter)
Input parameter	
absGroupId	It is the absolute group ID of the group object to be listed.
filter	A string input which denotes a directory search filter for the listMembers operation. A typical input of the form "ou=G1*" will returns all groups with name starting with the string G1. An input of null or "" implies no filter input and list all members under the given group.

Return Returns an array of GroupMembers that stores the names of the sub-groups and devices immediately under **absGroupId**.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **ParentNotFoundException** – Cannot locate group specified by absGroupId.
- **InvalidInputException** – See Table 6-2 for basic rules. In addition, input depth value can either be 1 or 0.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by remote API:

CommunicationException – General server access problem.

Example one

Based upon Figure 6-3 (b), the list member operation listMembers("/config/G1/G2",null) returns the following GroupMember array:

[("D1","/config/G1/G2/D1"),("D2","/config/G1/G2/D2"),
("G3","/config/G1/G2/G3"),("D4","/config/G1/G2/D4")].

Note the use of the notation ("D1","/config/G1/G2/D1") to denote a Element object that stores the element ID "D1" and the associated absolute element ID "/config/G1/G2/D1".

Example two:

A typical filter input of the form "(|(ou=G*)(IOSEventID=D*))" will list the immediate sub-group member with a name that start with the letter G and the immediate device elements with a name that start with a letter D.

List Groups

This operation returns the selected (immediate or descendant) members of the given group.

Method String[ ] listGroups (String absGroupId, int depth, String filter)

Input parameter

absGroupId Absolute group ID of the group object to be listed.

depth Set depth to 1 to list immediate groups under absGroupId or to 0 to list all (descendant) groups under absGroupId.

filter String input which denotes a directory search filter for the listGroups operation. A typical input of the form "ou=G1*" will returns all groups with name starting with the string G1. An input of null or " " implies no filtering.

Return Returns an array of absolute group ID (String) that denotes the sub-groups under **absGroupId**.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **ParentNotFoundException** – Cannot locate group specified by **absGroupId**.
- **InvalidInputException** – See Table 6-2 for basic rules. In addition, input depth value can either be 1 or 0.
- **MessageFormatException** – Problem with XML message syntax to be sent between the server and the remote API.

Additional exception that could be thrown by remote API:

CommunicationException – General server access problem.

Clone Members

This operation copies a list of group members and device members from the source group into the destination group (see Figure 6-4). Note that deep copy is used when copying a group object; all descendant groups are replicated and copied as well.

Method void cloneMembers (String srcAbsGroupId, String destAbsGroupId, GroupMember[] members)

Input parameter

srcAbsGroupId This is the absolute group ID of the source group.

destAbsGroupId This is the absolute group ID of the destination group.

members This is the GroupMember array that identifies the sub-group members and device members to be copied from the **srcAbsGroupId** into the **destAbsGroupId**.

Return None.

Exceptions

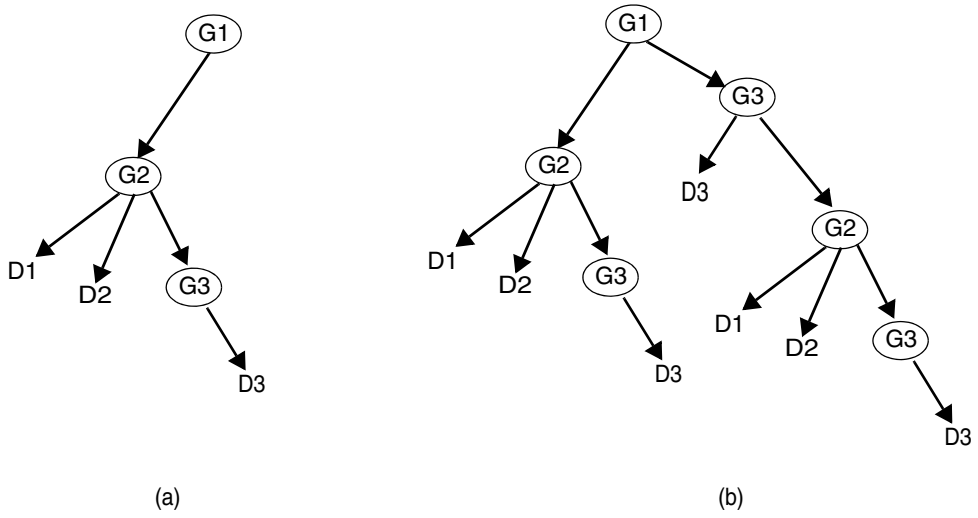
This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException**
  - See Table 6-2 for basic rules.
  - **srcAbsGroupId** and **destAbsGroupId** cannot be the same.
- **InvalidMemberException**
  - See Table 6-2 for basic rules.
  - The sub-group member and its descendent groups to be copied cannot result into an absolute group ID that violates the absolute-group-ID string-length restriction.
- **ParentNotFoundException** – Cannot locate group specified by **srcAbsGroupId** or **destAbsGroupId**.
- **MemberNotFoundException** – Member not found in group srcAbsGroupId.
- **MemberAlreadyExistsException** – Sub-group member or device member already existed in **destAbsGroupId**.
- **MessageFormatException** – Problem with message syntax.

Additional exception could be thrown by remote API:

CommunicationException – General server access problem.

Figure 6-4 Example of Clone Operation: (a) Before Clone (b) After Clone



Example:

Figure 6-4 shows two usages of the clone operation:

1. Clone a group: cloneMembers ("/config/G1/G2", "/config/G1", [G3]);
2. Clone a group hierarchy: cloneMembers ("/config/G1", "/config/G1/G3", [G2]);

Note Multiple occurrences of the same devices D1, D2, and D3 in the resulting tree are for the convenience of drawing. This does not imply the creation of multiple devices of the same name. They are only references to the same set of devices, D1, D2, and D3.

Clone All Members

This operation copies all group members from the source group into the destination group. Note that deep copy is used when copying a group object; all descendant groups are replicated and copied as well.

Method	void cloneAllMembers (String srcAbsGroupId, String destAbsGroupId)
Input parameter	
srcAbsGroupId	This is the absolute group ID of the source group.
destAbsGroupId	This is the absolute group ID of the destination group.
Return	None.
Exceptions	This method could throw one of the following exceptions: • **DatastoreAccessException** – Data-store connection problem or problem not described below. • **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process. • **InvalidInputException** – See Table 6-2 for basic rules. – **srcAbsGroupId** and **destAbsGroupId** cannot be the same. • **InvalidMemberException** – See Table 6-2 for basic rules. – The sub-group member and its descendent groups to be copied cannot result into an absolute group ID that violates the absolute-group-ID string-length restriction. • **ParentNotFoundException** – Cannot locate group specified by **absSrcGroupId** or **destSrcGroupId**. • **MemberAlreadyExistsException** – Sub-group member or device member already existed in **destAbsGroupId**. • **MessageFormatException** – Problem with message syntax. Additional exception that could be thrown by the remote API: **CommunicationException** – General server access problem.

Clone Groups

This operation copies the group members from the source group into the destination group, without any device member. Note that deep copy is used when copying a group object; all descendant groups are replicated and copied as well.

Method void cloneGroups (String srcAbsGroupId, String destAbsGroupId)

Input parameter

srcAbsGroupId This is the absolute group ID of the source group.

destAbsGroupId This is the absolute group ID of the destination group.

Return None.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process.
- **InvalidInputException**
  - See Table 6-2 for basic rules.
  - **srcAbsGroupId** and **destAbsGroupId** cannot be the same.
- **InvalidMemberException**
  - See Table 6-2 for basic rules.
  - The sub-group member and its descendent groups to be copied cannot result into an absolute group ID that violates the absolute-group-ID string-length restriction.
- **ParentNotFoundException** – Cannot locate group specified by **absSrcGroupId** or **destSrcGroupId**.
- **MemberAlreadyExistsException** – Sub-group member or device member already existed in **destAbsGroupId**.
- **MessageFormatException** – Problem with message syntax.

Additional exception that could be thrown by the remote API:

CommunicationException – General server access problem.

Rename Group

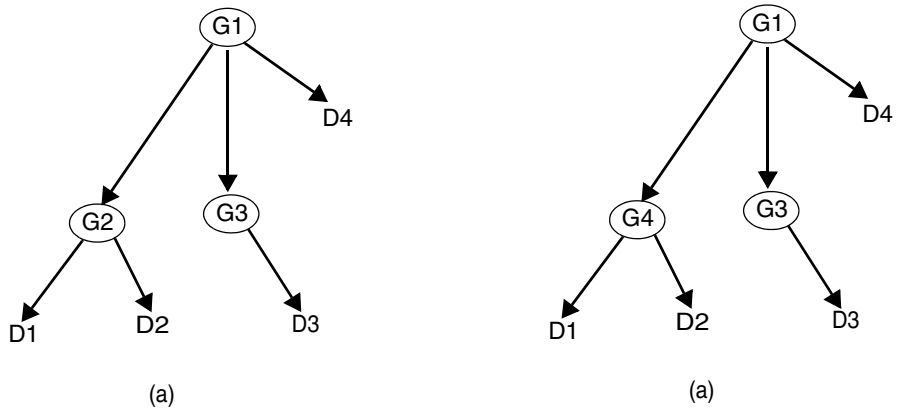
This operation renames an existing group to a new name.

Method void renameGroup (String absGroupId, String newGroupId)

Input parameter

absGroupId	This is the absolute group ID of the group to be renamed.
newGroupId	This is the new group name to be used.
Return	None.
Exceptions	This method could throw one of the following exceptions: • **DatastoreAccessException** – Data-store connection problem or problem not described below. • **OperationTimeoutException** – The current operation is not able to acquire a file lock within the time limit (for synchronizing data-store access), because other administration operation is currently in process. • **InvalidInputException** – See Table 6-2 for basic rules. – After renaming, the resulting absolute group ID cannot violate the absolute-group-ID string-length restriction. – **newGroupId** cannot contain a slash "/", because it is used as a delimiter for group hierarchy. • **MemberAlreadyExistsException** – Member already existed; in other words, there exists a group with name **newGroupId** in the same parent group of **absGroupId**. • **ParentNotFoundException** – Cannot locate group specified by **absGroupId**. • **MessageFormatException** – Problem with message syntax. Additional exception that can be thrown by the remote API: **CommunicationException** – General server access problem.

Figure 6-5 Rename Group /config/G1/G2



Example:

Given Figure 6-5 (a), renaming group /config/G1/G2 to /config/G1/G3 using the renameGroup operation renameGroup ("/config/G1/G2","G3") will throw a MemberAlreadyExistsException because group /config/G1/G3 already existed.

Figure 6-5 (b) illustrates the result of renaming /config/G1/G2 into /config/G1/G4 using renameGroup("/config/G1/G2","G4").

isMember

This operation checks if the given member is a direct member of the given group, returning TRUE if it is a member or FALSE otherwise.

Method	Boolean isMember (String absGroupId, GroupMember member)
Input parameter	
absGroupId	This is the absolute group ID of the given group.
member	This is the member to be checked.
Return	A boolean value indicates whether the given member is an immediate member of group absGroupId.
Exceptions	This method could throw one of the following exceptions: • **DatastoreAccessException** – Data-store connection problem or problem not described below. • **InvalidInputException** – See Table 6-2 for basic rules. • **ParentNotFoundException** – Cannot locate group specified by **absGroupId**. • **MessageFormatException** – Problem with message syntax. Additional exception that can be thrown by the remote API: **CommunicationException** – General server access problem.

List Parents

This method returns all the parent groups that are direct parents of the given device.

Method	String[ ] listParents (String device)
Input parameter	
device	This is the device ID for which the direct parents should be identified.
Return	An array of absolute group ID that are immediate parents of the given device.

Exceptions This method could throw one of the following exceptions:

- **DatastoreAccessException** – Data-store connection problem or problem not described below.
- **InvalidInputException** – See Table 6-2 for basic rules.
- **MemberNotFoundException** – Given device does not exist.
- **MessageFormatException** – Problem with message syntax.

Additional exception that can be thrown by remote API:

CommunicationException – General server access problem.

Example:

Given Figure 6-5 (b), the operation listParents("D3") returns the following string array:

["/config/G1/G2/G3","/config/G1/G3","/config/G1/G3/G2/G3"]

Set Notification

This method provides the client application the capability of stopping and re-starting subsequent group administration operations from generating change notification events.

In cases where the client application is expected to perform a great number of groups and devices re-arrangement operations, the client application could choose to stop notification temporarily (using setNotification(FALSE)), carry out the operations, and then re-enable notification (using setNotification(TRUE)) after. This could reduce the event traffic and avoid handling of a large number of notification events.

When re-enabling notification, this method sends a **global-change** notification to all subscribing applications, asking them to re-load all its subject mappings to the ones defined by the latest grouping and namespace definitions.

Method void setNotification (Boolean state)

Input parameter

state A boolean indicates whether to turn on or turn off the sending of notification event for the subsequent change operation.

Return None.

Exceptions Exception that can be thrown:

MessageFormatException – Problem with message syntax.

Exception that can be thrown by the remote API:

CommunicationException – General server access problem.

Partial-complete Operation Upon Failure

Most of the Group administration APIs involve manipulation of multiple members, such as move multiple members, delete multiple members, and so forth. The operation behavior upon failure is defined as follows: The operation terminates as soon as an error is encountered and the **getInfo()** method defined in **OperationFailedException**, and inherited by the related exceptions (see "C++ Error Codes" section on page 6-46), returns a **MemberElementInfo** object, that contains the following information:

```
public class MemberElementInfo
{
  public GroupMember getFailedElement()
  public GroupMember[] getCompletedElements()
  public GroupMember[] getRemainingElements()
}
```

- Failure member ID which causes the failure.
- Completed member IDs which have successfully passed the operation.
- Remaining members IDs which have not been done by the operation.

These are information collected when the operation fails in the middle of processing the members array. However, the same object is also used to report the absolute group ID when a **parentNotFoundException** is thrown; in which case, the failure member object ID is the absolute group ID of the parent group not found.

The following exceptions always have a **MemberElementInfo** object defined:

- MemberAlreadyExistsException
- MemberNotFoundException
- InvalidMemberException
- ParentNotFoundException

The following exceptions may or may not have a MemberElementInfo object defined:

- InvalidOperationException
- DatastoreAccessException.

These are exceptions that can occur before or in the middle of processing the member lists.

Notification API

The Data-Change Notification interface provides a way for client applications to be notified of data changes in the data store with regard to namespace and grouping.

Notification happens through a callback that the application registers with the interface. Within the callback, the application can perform the tasks necessary to resolve its set of subjects again.

Notification client registers to the group-change and namespace-change notification service, respectively, by instantiating a **GroupChangeNotificaton** object and a **NamespaceChangeNotification** object.

Class GroupChangeNotification

GroupChangeNotification

This is the constructor of the **GroupChangeNofitiation** class.

Method	GroupChangeNotification (String namespace, NotificationHandler callback, Object callbackArg)
Input parameters	
namespace	Namespace Id of interest.
callback	Callback object which implements NotificationHandler with the onNotification method defined.
callbackArg	Object to be passed to the onNotification method of the callback object.
Return	None.

startNotification

This method setups the notification transport and creates the listener on the group-change-notification event (in the form *cisco.mgmt.cns.group.notify.<namespace>*) for the given namespace.

Method	int startNotification ()
Input parameters	None.
Exception	This methods throws NotificationException upon listener creation failure.

stopNotification

This method terminates the listener previously created using startNotification.

Method	stopNotification ()
Input parameters	None.
Exception	This methods throws NotificationException upon listener termination failure.

Class NamespaceChangeNotification

NamespaceChangeNotification

This is the constructor of the NamespaceChangeNofitiation class.

Method	NamespaceChangeNotification (String namespace, NotificationHandler callback, Object callbackArg)
Input parameters	
namespace	Namespace Id of interest.
callback	Callback object which implements NotificationHandler with the onNotification method defined.
callbackArg	Object to be passed to the onNotification method of the callback object.
Return	None.

startNotification

This method setups the notification transport and creates the listener on the namespace-change-notification (in the form *cisco.mgmt.cns.namespace.notify.<namespace>)* event for the given namespace.

Method	int startNotification ()
Input parameters	None.
Exception	This methods throws NotificationException upon listener creation failure.

stopNotification

This method terminates the listener previously created using startNotification.

Method	stopNotification ()
Input parameters	None.
Exception	This methods throws NotificationException upon listener termination failure.

Class NotificationHandler

This class contains the **onNotification** method that the callback object must implement. When the DataChangeNotification object detects a change notification, it creates either a GroupChangeInfo object or a NamespaceChangeInfo object, which captures the change information, and passes it onto the callback handler.

Method	void onNotification (DataChangeInfo dataChangeInfo, String namespace, Object callbackArg)
Input parameters	
dataChangeInfo	For GroupChangeNotification callback object, this is the GroupChangeInfo object that is passed to the onNotification callback handler. See "Class GroupChangeInfo" section on page 6-36 for details. For NamespaceChangeNotification callback object, this is the NamespaceChangeInfo object that is passed to the onNotification callback handler. See "Class NamespaceChangeInfo" section on page 6-39 for details.
namespace	Namespace Id of the group-change or namespace-change event that triggers this callback.
callback	Callback object which implements NotificationHandler with the onNotification method defined.
callbackArg	Context object input when constructing the GroupChangeNotification or the NamespaceChangeNotification object.
Exception	This method should throw NotificationHandlerException to report any callback client failure.

Class DataChangeInfo

This is the base class of the two derived classes GroupChangeInfo and NamespaceChangeInfo. It defines the following public constants for identifying the group or namespace operation that is related to change information object:

ADD_MEMBERS
DEL_MEMBERS
DEL_ALL_MEMBERS
MOVE_MEMBERS
CLONE_MEMBERS
CLONE_ALL_MEMBERS
RENAME_GROUP
ENABLE_GROUP_NOTIFICATION
DISABLE_GROUP_NOTIFICATION

ADD_NAMESPACE
DEL_NAMESPACE
CLONE_NAMESPACE
ADD_SUBJECT
DEL_SUBJECT
ADD_MAP
DEL_MAP
ENABLE_NAMESPACE_NOTIFICATION
DISABLE_NAMESPACE_NOTIFICATION

Note The method **getOpType()** can be used to retrieve the operation identifier of the GroupChangeInfo and NamespaceChangeInfo objects.

Class GroupChangeInfo

Group Change Information is a data object created by the DataChangeNotification object upon arrival of the notification event, for reporting changes made by a group administration operation. It is used for conveying group change information to the client's NotificationCallback object.

This class provides six member functions for the callback function to retrieve the group change information.

int getOpType()

Returns the operation type identifier

String getSourceGroupId()

This is the source absolute group ID that specifies the group on which the operation is performed. See "Grouping rules" section on page 6-14 for the definition of absolute group ID.

String getDestinationGroupId()

This is the destination absolute group ID that denotes the destination group for a MOVE or CLONE operation. See "Grouping rules" section on page 6-14 for the definition of absolute group ID. In addition, it is also used to return the new Group Id in the rename group operation, e.g., renameGroup(absGroupId, newGroupId).

GroupMember[] getMembers()

This is an array of GroupMember objects specified in the group administration operation. Each object contains a group ID or a device ID. See "GroupMember" section on page 6-17 for the definition of GroupMember object.

String[] getGroups()

Returns an array of affected absolute group IDs for which the application should re-acquire all its mapping from NSM.

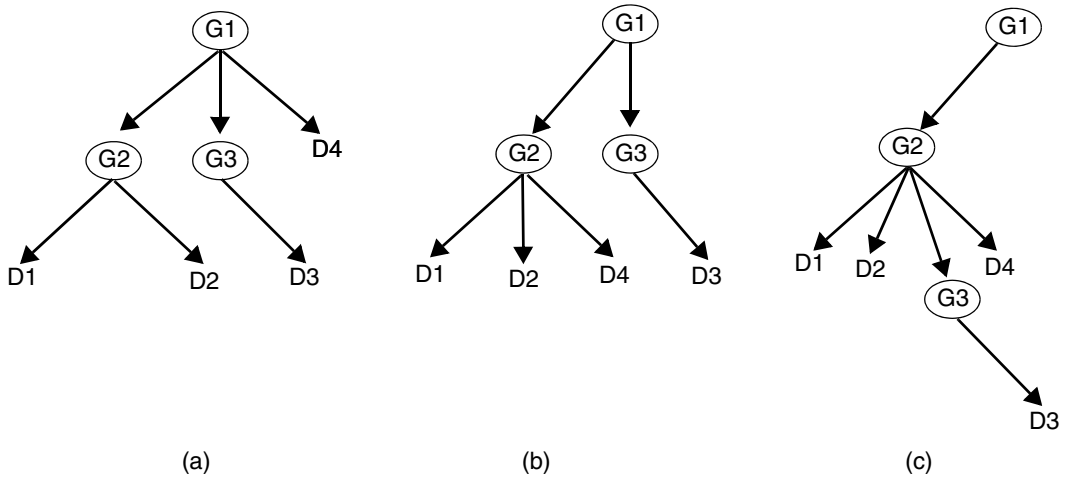
String[] getElements()

Returns an array of affected device IDs for which the application should re-acquire all its mapping from NSM.

The returned operation type, source group ID, destination group ID, and group member array together provide the details of the original operation. For group operation that involves only one operand, such as deleteAllMembers(absGroupId), the destination group ID and group member array will be set to NULL. For operation that involves two operands such as deleteMember(absGroupId, memberArray), the destination group Id will be set to NULL.

Application which requires the latest mapping information should check the list of affected (absolute) group IDs and device IDs returned and re-acquired all the associated mappings from NSM.

Figure 6-6 Example of move members operation in group administration API



Example one:

The move operation **moveMembers("/config/G1", "/config/G1/G2", [D4]);** will move device D4 from group /config/G1 to group /config/G1/G2, as shown in Figure 6-6 (a) and (b).

After successfully completing the move operation, the following information are returned by the associated GroupChangeInfo object:

getOpType()	MOVE_MEMBER
getSourceGroupId()	"/config/G1"
getDestinationGroupId()	"/config/G1/G2"

getMembers()	[D4]
getGroups()	[“/config/G1”, “/config/G1/G2”]
getElements()	[“D4”]

The first four returned parameters describe the original move operation.

Function getGroups returns the list of affected groups, i.e., NSM client application which has previously subscribed to event using any group on the affected group lists should re-subscribe again. Due to the move of device D4, the resulting group-name qualified subscribe map have been changed. See “Mapping Scenarios” section on page 6-43 for the generation of the group-name qualified subscribe map.

Similarly, application which has previously subscribed to event using any device on the affected element array (returned by getElements) should re-subscribe again. The move of device D4 has also changed the resulting device-name qualified subscribe map. See “Mapping Scenarios” section on page 6-43 for the generation of the device-name qualified subscribe map.

Example two:

The move operation **moveMembers(“/config/G1”,”/config/G1/G2”, [G3]);** will move group G3 from group /config/G1 to group /config/G1/G2, as shown in **Figure 5-1** (b) and (c).

Note that device D3 is implicitly affected due to a change in the ancestor group. After successfully completing the move operation, the following GroupChangeInfo object will be created:

getOpType()	MOVE_MEMBER
getSourceGroupId()	“/config/G1”
getDestinationGroupId()	“/config/G1/G2”
getMembers()	[G3]
getGroups()	[“/config/G1”, “/config/G1/G2”, “/config/G1/G2/G3”]
getElements()	[“D3”]

Example three:

The previous two move operations can in fact be combined in one single move:

moveMembers("/config/G1", "/config/G1/G2", [D4,G3])

Upon completion of the move, the following GroupChangeInfo object will be created:

getOpType()	MOVE_MEMBER
getSourceGroupId()	"/config/G1"
getDestinationGroupId()	"/config/G1/G2"
getMembers()	[D4,G3]
getGroups()	["/config/G1", "/config/G1/G2", "/config/G1/G2/G3"]
getElements()	["D3", "D4"]

Class NamespaceChangeInfo

This is an information object used to convey namespace change information to the NSM client. It is created by the NamespaceChangeNotification object when parsing the information received from the namespace change event, which in turn, is generated by the Namespace Administration API.

The following seven member function can be used to retrieve information stored in NamespaceChangeInfo object:

int getOpType()	Returns the operation type identifier.
public String getNamespaceId ()	Returns the IDs of the namespace that was changed.
public String getSubject ()	Returns the subject that was added, deleted, or modified.
public int getMapType ()	Returns the mapping type, such as PUBLISH or SUBSCRIBE.
public int getResolveMode ()	Returns the resolve mode, such as ALGORITHMIC or NON_ALGORITHMIC.
public String getCloneId ()	Returns the new namespace ID for the cloneNamespace operation.
public String[] getMapping ()	Returns the array of associated mapping.

Dynamic Grouping of Devices

When grouping information in the data store is modified through the Administrative interface (see "Group Administration API" section on page 6-14), a trigger is published on the Integration Bus.

An application that requires notification can call the data change notification interface, thereby starting a listener on the trigger subject. The application upon receiving the notification can then call the Namespace Mapper client API another time to resolve the desired subject.

Any application that subscribes to mapped subjects must resolve again upon receiving the change notification because their listeners would otherwise become outdated. The application shall then create listeners on the newly returned subjects after deleting the old listeners.

Applications that cache or save the results of the resolve method should also call **resolve** again if a change occurs in the data store.

Synchronization Between Publishers and Subscribers

There is a potential timing mismatch with the interaction discussed above. Suppose that a data change notification is sent out because of a grouping information change. It takes a finite amount of time for a subscribing client application to process the notification and update its listeners. While this is going on, if a publishing application calls the Namespace Mapper to resolve a given subject, it will be returned the set of new mappings. If the new subjects are published on the Integration Bus before the subscriber updates the listeners, the published events will be lost.

Hence there is a window during which the subscriber could lose events. The duration of the window could vary depending on the number of listeners to be updated by the subscriber. It is suggested that group and namespace administrations be done within a system maintenance window and sufficient waiting time be given, after the changes, before resuming the regular operations.

NSM Clients in Cisco Configuration Engine

There are three applications in Cisco Configuration Engine using the Namespace mapper:

- Event Gateway
- Intelligent Modular Gateway (IMGW)
- Configuration Server

Event Gateway

The Event Gateway acts as a proxy for agent-enabled devices and calls the NSM client API to resolve the subjects passed by devices connected to it. Since it subscribes to the mapped subjects, the Event Gateway should act on the change notification and update its listeners.

Intelligent Modular Gateway

The IMGW process acts in the same way as the Event Gateway except that it acts on behalf of devices that do not have agents running. It also subscribes to mapped subjects and hence should handle the change notification and update its listeners and caches.

Configuration Server

The Configuration calls the NSM client API to resolve subjects and publishes on them. It also listens to event ID change events and status events from the device. So it must also call the Data Change Notification Interface and update its listeners.

NSM Support for Hierarchical Groups

The earlier Namespace Mapper implementation did not support the concept of a group being a member of another group. This is a new feature of Cisco Configuration Engine.

With the introduction of hierarchical groups, where a group can be a member of another group, you can better organize your devices including moving a group of devices from one hierarchy to the other.

Model Change

The design of the hierarchical grouping model has lead to the following namespace (or application) and group relationship:

- Each namespace is required to define its own group hierarchies.
- If a group hierarchy is to be shared between different namespaces, the structure must be copied from one namespace to the other. Don't forget to synchronize the group structure in both namespaces.

A Group has two properties in this model:

- Each group will end up having one namespace association.
- Groups that lie in the same hierarchy are always assigned to the same namespace.

Advantages

This model has the following advantages:

- The overall relationship between hierarchy grouping and namespace is much clearer and easier to maintain by the user. The users will always have a clear picture of the group hierarchy for the given namespace.
- The generation of the notification events for the grouping API is much simpler. It no longer requires an algorithm to deduce the affected namespace.

Disadvantage

A common grouping structure that is mean to be shared among multiple namespaces must be copied into individual namespace; changes made in one will not be reflected in the others. However, you have the choice of re-copying the group hierarchy.

Schema Changes

- The directory class **groupOfNames** is used when creating the single-level grouping structure in Cisco Configuration Engine. Its support for hierarchical groups is directory specific and is not generally available. In order to support external directory developed by different vendors, the new group class **cnsGroup** is defined. It is a sub-class of the fundamental container class **OrganizationUnit** and contains one multi-value reference attribute cnsMember. cnsMember is a reference which points to the device member of the group.
- The namespace and group association (the seeAlso attributes in the namespace container and in the group container which points to each other) is no longer required.
- All group containers will be stored inside the associated namespace containers.

Mapping Scenarios

This section describes the mapping scenarios that are supported by NSM in hierarchical grouping.

Example

Figure 6-7 shows an example of hierarchical groups. Table 6-1 lists the sample mappings stored in namespaces NS1.

Figure 6-7 Example Of Hierarchical Grouping For namespace NS1

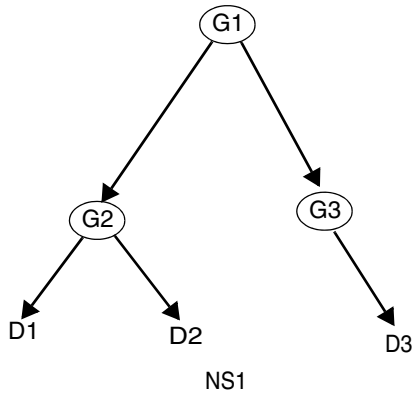


Table 6-1 NS1 Mapping Table

	NS1
Subject	*cisco.mgmt.cns.config.load*
Subscriber map	*cisco.mgmt.cns.config.load.vpn1*
Publisher map	*cisco.mgmt.cns.config.load.vpn1*

Terminology

Given below are two sets of terminologies respectively for describing the hierarchical-grouping elements and the resulting maps.

- **Parent** of a group G (or a device D) refers to the direct super group that contains group G (or device D). For example, in Figure 6-7, the parent group of device D3 is G3 while the parent group of G3 is G1.
- **Ancestor(s)** of a group G (or a device D) refers to the direct and indirect super groups of G (or device D) in the hierarchy. For example, the ancestor group of group G3 is G1 while the ancestor groups of device D3 are G3 and G1.
- **Children** of group G refer to the direct sub-groups and device members of group G. For example, the children of group G1 are groups G2 and G3 while the child of G3 is device D3.
- **Descendant** of group G refers to the direct and indirect sub-groups and device members of group G. For example, the descendant of group G1 are groups G2 and G3 and devices D1, D2, and D3.

- **Mapped** events refer to the resulting events that NSM retrieved from the namespace. For example, the entries listed in the subscriber map and publisher map in Table 6-1 are the mapped events.
- **Device-name qualified** events refer to the resulting events obtained by appending a device ID to the end of the **mapped** event.
- **Group-name qualified** events refer to the resulting events obtained by appending a **absolute** group ID to the end of the **mapped** event. Note the usage of absolute group ID here, because it is required to uniquely identify a group in hierarchical grouping.

NSM Mapping Rules

- NSM client must supply the following parameters when querying NSM for a mapping: Namespace identifier, event identifier, mapping type (PUBLISH or SUBSCRIBE), and a device or group identifier.
- Given an absolute group ID and namespace ID, NSM retrieves the mapped events only if the group can be found in the given namespace; otherwise, it returns an empty mapping. That is, when NSM client wants to retrieve the mapping for a group event, it must specify a group that belongs to the given namespace.

  When it wants to retrieve the mapping for a device event, one of the device's parent groups must belong to the input namespace.

Mapping Scenarios

Eight mapping scenarios are identified and divided into two categories, non-algorithmic and algorithmic. Qualified events are returned in algorithmic mode and mapped events in non-algorithmic mode. The next two sections present the eight scenarios. The next two sections present eight scenarios based on the example shown in Figure 6-7.

Algorithmic Mode

Scenario One:

Mapping of a group-publish event returns the group-name qualified events.

Example:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for publishing a load event on group /NS1/G1.

NSM retrieves the mapped event *cisco.mgmt.cns.config.load.vpn1* and returns the group-name qualified event *cisco.mgmt.cns.config.load.vpn1./NS1/G1*.

Scenario Two:

Mapping of a device-publish event returns the device-name qualified events.

Example:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for publishing a load event on device D1.

NSM checks if D1 belongs to an ancestor group that is in the input namespace. Since group /NS1/G2 belongs to namespace NS1, NSM retrieves the mapped event *cisco.mgmt.cns.config.load.vpn1* and returns the device-name qualified event *cisco.mgmt.cns.config.load.vpn1.D1*. If D1 is not related to the namespace NS1, NSM would return an empty mapping.

Scenario Three:

Mapping of a group-subscribe event returns both the group-name and device-name qualified events.

Example One:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for subscribing to the load event at group /NS1/G1, say, for monitoring purpose.

NSM retrieves the mapped events from NS1 and identifies all /NS1/G1's descendants. Qualified events are then constructed using the namespace associated group names and device names.

NSM returns the following qualified events:

cisco.mgmt.cns.config.load.vpn1./NS1/G1

cisco.mgmt.cns.config.load.vpn1./NS1/G1/G2

cisco.mgmt.cns.config.load.vpn1./NS1/G1/G3

cisco.mgmt.cns.config.load.vpn1.D1

cisco.mgmt.cns.config.load.vpn1.D2

cisco.mgmt.cns.config.load.vpn1.D3

As a result, the NSM client will catch any load event published at one of the groups /NS1/G1, /NS1/G1/G2 and /NS1/G1/G3 or one of the devices D1, D2, and D3.

Example Two:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS2 for subscribing to the load event at group /NS1/G1.

NSM returns an empty mapping because /NS1/G1 is not associated with namespace NS2.

Scenario Four:

Mapping of a device-subscribe event returns both group-name and device-name qualified events.

Example One:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for subscribing to a load event at device D1.

NSM checks if D1 belongs to a parent group which belongs to the given namespace. If a group is found, NSM retrieves the mapped event *cisco.mgmt.cns.config.load.vpn1* and qualifies it using the device ID and the ID of the group ancestors associated with the given namespace.

NSM returns the following qualified events:

cisco.mgmt.cns.config.load.vpn1./NS1/G1

cisco.mgmt.cns.config.load.vpn1./NS1/G1/G2

cisco.mgmt.cns.config.load.vpn1.D1

Example Two:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for subscribing to a load event at device D3.

NSM checks if D3 belongs to a group that belongs to the input namespace. If there is an association found, NSM retrieves the mapped event *cisco.mgmt.cns.config*.

NSM returns the following qualified events:

cisco.mgmt.cns.config.load.vpn1./NS1/G1

cisco.mgmt.cns.config.load.vpn1./NS1/G1/G3

cisco.mgmt.cns.config.load.vpn1.D3

Non-algorithmic Mode

Only mapped events are returned in non-algorithmic mode.

Scenario One:

Mapping of a group-publish event returns the mapped events.

Example One:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for publishing a load event on group /NS1/G1.

NSM retrieves and returns *cisco.mgmt.cns.config.load.vpn1*, the mapped event.

Example Two:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS2 for publishing a load event on group /NS1/G1.

NSM returns an empty mapping because /NS1/G1 is not associated with namespace NS2.

Scenario Two:

Mapping of a device-publish event returns the mapped events.

Example:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for publishing a load event on device D1.

NSM checks if D1 belongs to a group that belongs to the input namespace. Since /NS1/G2 belongs to NS1, NSM retrieves and returns the mapped event *cisco.mgmt.cns.config.load.vpn1*.

Scenario Three:

Mapping of a group-subscribe event returns the mapped events.

Example One:

NSM client ask NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for subscribing to a load event on group /NS1/G1.

NSM retrieves and returns *cisco.mgmt.cns.config.load.vpn1*, the mapped event.

Example Two:

NSM client ask NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS2 for subscribing to a load event on group /NS1/G1.

NSM returns an empty mapping because /NS1/G1 is not in namespace NS2.

Scenario Four:

Mapping of a device-subscribe event returns the mapped events.

Example One:

NSM client asks NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS1 for subscribing to a load event at device D1.

NSM checks if D1 belongs to a group that is in the given namespace. Since /NS1/G1/G2 is found, NSM retrieves and returns the mapped event *cisco.mgmt.cns.config.load.vpn1*.

Example Two:

NSM client ask NSM to resolve the subject *cisco.mgmt.cns.config.load* using namespace NS2 for subscribing to a load event at device D3.

NSM returns an empty mapping because there is no association between the parent groups of D3 and NS2.

C++ Error Codes

Java API implementation will make use of the exception-handling approach for reporting errors while C++ API implementation will adopt the return-code approach. This is so, because C++ compiler does not enforce exception handling in compile-time, but Java does. The error codes defined for C++ and the exceptions defined for Java are one-to-one correspondent; they are shown in Table 6-2.

Indentation within the exceptions column shows the hierarchy of exception inheritance. For example, class **MemberAlreadyExistsException** (with error code 152) is derived from the base class **OperationException**, which in turn is derived from GroupAdminException.

Table 6-2 Error Codes

Error code	Exception	Description
10	InvalidInputException	Any incorrect inputs, such as the followings: • Input argument of String type cannot be NULL or equal to “” (empty string). • For array input, the array cannot be NULL or of size 0. • Input argument absolute group Id (of String type) cannot be NULL or “” (empty string). In addition, absolute group Id must start with a slash “/” and does not end with a slash. However, “/” is a special case of valid input, denoting the root. There is a configurable-length restriction imposed on the length of absolute group Id, for resource control reason. Default length will be defined in a configuration file. • Input argument such as member (of type GroupMember) and members (of type GroupMember[]) cannot be NULL. • For Group Admin Remote API, the invalid character set for group id and member id is {0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0 x29,0x2A,0x2B,0x2C,0x3A,0x3B,0x3C, 0x3D, 0x3E,0x3F,0x40,0x5B,0x5C,0x5D,0x5E, 0x60} • For Namespace Admin Remote API, the invalid character set for namespace id, subject and subject mapping is {0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0 x29,0x2A,0x2B,0x2C,0x2F,0x3A,0x3B,0x3C,0 x3D,0x3E,0x3F,0x40,0x5B,0x5C,0x5D,0x5E,0 x60}
11	CommunicationException	HTTP or servlet connection problem.
12	OperationTimeoutException	Fail to acquire a file lock within the given time limit and number of retrials.
13	MessageFormatException	Problem with the XML message syntax to be sent between the server and the remote API.
100	General operation	SUCCESS.

Table 6-2 Error Codes (continued)

Error code	Exception	Description
125	NotificationAdminException	General Notification administration exception.
150	GroupAdminException	General group administration problem not described below:
151	OperationFailedException	General operation failure
152	MemberAlreadyExistsException	Member already existed.
153	MemberNotFoundException	Member not found or When device does not exist in listParents(device).
154	InvalidMemberException	Given an array **members** of GroupMember objects, • if one of members' array entry is NULL, or • if a GroupMember object's ID is an empty string or NULL, or in case of a group ID, the resulting absolute group ID violates string length restriction described in InvalidInputException. • In addition, group ID or device ID cannot contain a slash, because it is used as a delimiter in the absolute group name and a absolute group name is used when generating a group name qualified event.
155	ParentNotFoundException	Cannot locate a group, specified by the given absolute group ID, in the hierarchy.
156	InvalidOperationException	An invalid operation as described in moveMembers.
157	DatastoreAccessException	Data store connection problem or problem not described above.
200	NamespaceAdminException	
201	OperationFailedException	General operation failure
202	NamespaceAlreadyExistsException	Namespace already exists when adding a namespace, etc.
203	NamespaceNotFoundException	Namespace not found when deleting a namespace or query a namespace, etc.
204	MappingAlreadyExistsException	Mapping already exists when adding a mapping for a specific subject.
206	MappingNotFoundException	Mapping not found when deleting a mapping for a specific subject.
207	InvalidMappingException	NULL in subjectMapping array entry. Empty String in subjectMapping array entry.

Table 6-2 Error Codes (continued)

Error code	Exception	Description
208	SubjectAlreadyExistsException	Subject already exists when adding a subject for a specific namespace.
209	SubjectNotFoundException	Subject not found when deleting a mapping for a specific subject.
211	GroupFoundException	When client tries to delete the Namespace while there are groups defined in the namespace.
212	DatastoreAccessException	Data store connection problem or problem not described above.

ErrorInfo

Instead of returning an error-code directly in the C++ implementation, each API sets the error code in an **ErrorInfo** object. The client should check the error status before executing the next administration API because the error code would have been changed to the status of the next API after. It can be retrieved using the **getErrorInfo()** method defined in the C++ version of the administration API.

An error information object is created per (group or namespace) administration object per thread, using the ACE thread specific storage construct. Each client thread that has a handle to the same administration object will have its own error information object. That is, the administration object can be used by multiple threads and is thread safe.

For group administration API, the **getMemberElementInfo** in the **ErrorInfo** object provides a **MemberElementInfo** object whenever additional information regarding the failure is available.

Shown below is the structure of the **ErrorInfo** class:

```
class ErrorInfo
{
    friend class GroupAdminRemote;
    friend class DatachangeNotification;

public:

    void getMemberElementInfo(ReturnObject<MemberElementInfo>& result);
    int  getErrorCode();
    char*  getDescription();

private:
    ErrorInfo();

    // define copy constructor for performing a deep copy.
    ErrorInfo(const ErrorInfo& errorInfo);

    ~ErrorInfo();

    ErrorInfo (int statusCode, char *description);
    ErrorInfo (int statusCode, char *description, boolean transient);
    ErrorInfo (int statusCode, char *description, boolean transient, char **ids);

    // Group's member info
    ErrorInfo (int statusCode, char *description, MemberElementInfo* change);

    // disallow client copying
```

```
    ErrorInfo(const ErrorInfo& errorInfo);

    // free memory allocated in the library for data members.
    freespace();

private:
    int _statusCode;
    ACE_CString description;
    boolean transient;
    ACE_Array<ACE_CString> _ids;
    MemberElementInfo* m_memberInfo;
};
```

MemberElementInfo

```
class MemberElementInfo
{
    friend class GroupAdminRemote;

public:
    GroupMember* getFailedElement();
    GroupMember** getCompletedElements(int& memberCnt);
    GroupMember** getRemainingElements(int& memberCnt);

private:
    MemberElementInfo();

    // define copy constructor for deep copy
    MemberElementInfo(const&  MemberElementInfo memberEle);

    MemberElementInfo(GroupMember* failedEle, GroupMember** completedEle,
    GroupMember** remainingEle);
    MemberElementInfo(const MemberElemenInfo& memberEle);
    ~MemberElementInfo();


    GroupMember* m_failedElement;
    GroupMember** m_completedElements;
    GroupMember** m_remainingElements;
}
```

ReturnObject

When retrieving the MemberElementInfo object using the get methods of **ErrorInfo**, the client must declare a result container using the template ReturnObject; for example, **ReturnObject<MemberElementInfo> resultStore**. Afterwards, the client uses the **getValues** method of **resultStore** to get a reference of the information object.

This is to work around a space allocation strategy specific to Microsoft libraries: library allocated space cannot be free by the client applications, using the regular delete call. The purpose of **ReturnObject** is to wrap around the library allocated objects so that the **ReturnObject** destructor (invoked when deleted by the client) can invoke the appropriate library function to destroy the library allocated objects. As a result, all data returned by the API are data created in the library space and cannot be deleted by the client applications, unless a container is defined using the ReturnObject template.

```
template <class T>
class ReturnObject
{
```

```
private:
    T[]  m_val;
    int  m_size;

    void  setValues(T[]  m_val, int count);
    void  freespace();

public:
    ReturnObject();

    // define copy constructor for deep copy.
    ReturnObject(const ReturnObject& ReturnObject);

    ~ReturnObject();

    T*  getValues(int& count);
}
```

Namespace Administration API Reference

C++ Version

```
/* C++ Interface Namespace Admin */
  enum MappingType {
        INVALID_MAPPING = -1,
        PUBLISH,
        SUBSCRIBE
  };

  enum ResolveMode {
        INVALID_MODE = -1,
        NON_ALGORITHMIC,
        ALGORITHMIC
  };

  enum OperationType {
        INVALID_OP = -1,
        ADD_NAMESPACE_OP = 1,
        DEL_NAMESPACE_OP,
        CLONE_NAMESPACE_OP,
        LIST_NAMESPACES_OP,
        ADD_SUBJECT_OP,
        DEL_SUBJECT_OP,
        LIST_SUBJECTS_OP,
        ADD_MAPPING_OP,
        DEL_MAPPING_OP,
        LIST_MAPPINGS_OP,
        GET_RESOLVE_MODE_OP,
        SET_NOTIFICATION_OP
  };

enum {SUCCESS, FAILED};

class NamespaceAdmin
{
  virtual int addNamespace( const char* namespaceId ) = 0;
```

```
  virtual int delNamespace( const char* namespaceId ) = 0;

  virtual int listNamespaces(
                ReturnStrArray& result,
                const char* filter = 0) = 0;

  virtual int cloneNamespace(
                const char* namespaceId,
                const char* newNamespaceId ) = 0;

  virtual int addSubject(
                const char* namespaceId,
                const char* subject) = 0;

  virtual int delSubject(
                const char* namespaceId,
                const char* subject) = 0;

  virtual int listSubjects(
                const char* namespaceId,
                ReturnStrArray& result,
                const char* filter = 0) = 0;

  virtual int addSubjectMapping(
                const char* namespaceId,
                MappingType mapType,
                const char* subject,
                const char** subjectMapping,
                int size,
                ResolveMode mode) = 0;

  virtual int delSubjectMapping(
                const char* namespaceId,
                MappingType mapType,
                const char* subject) = 0;

  virtual int listSubjectMappings(
                const char* namespaceId,
                MappingType mapType,
                const char* subject,
                ReturnStrArray& result) = 0;

  virtual int getResolveMode(
                const char* namespaceId,
                MappingType mapType,
                const char* subject,
                int &resolveMode) = 0;

  virtual int setNotification(boolean state) = 0;

  ErrorInfo* getErrorInfo();

}

/* Factory class for NamespaceAdmin */
class NamespaceAdminFactory
{
  public:
    NamespaceAdminFactory();
    NamespaceAdminFactory(const Properties& prop);
    NamespaceAdminFactory(const char* propFile);

    ~NamespaceAdminFactory();
```

```
    NamespaceAdmin* createAdmin();
    void deleteAdmin(NamespaceAdmin* admin);
}

class Properties
{
  public:
    Properties();
    Properties(const Properties& property);
    Properties(const char* filename);
    ~Properties();

    Properties& operator=(const Properties& property);

    void setProperty(const char* name, const char* value);

    // caller must delete the return char* with releaseString
    char* getProperty(const char* name) const;

    void releaseString(char* s) const;

    void delProperty(const char* name);
    void listAllPropertyNames(ReturnStrArray& names, int& count) const;

    static char *getPropFile(const char *prop_file);
}
```

Java Version

```
/* Java Interface Namespace Admin */

public interface NamespaceAdmin
{
void  addNamespace (String  namespaceId) throws
           DatastoreAccessException,
           NamespaceAlreadyExistsException,
           InvalidInputException,
           CommunicationException,
           MessageFormatException,
           OperationTimeoutException;

void  delNamespace (String  namespaceId) throws
           DatastoreAccessException,
           InvalidInputException,
           NamespaceNotFoundException,
           CommunicationException,
           SubjectAlreadyExistsException,
           GroupExistsException,
           MessageFormatException,
           OperationTimeoutException;

void  cloneNamespace(String namespaceId, String newNamespaceId) throws
           DatastoreAccessException,
           InvalidInputException,
           NamespaceNotFoundException,
           NamespaceAlreadyExistsException,
           CommunicationException,
           MessageFormatException,
           OperationTimeoutException;


String[] listNamespaces() throws
```

```
            DatastoreAccessException,
            MessageFormatException,
            CommunicationException;

String[] listNamespaces(String filter) throws
            DatastoreAccessException,
            MessageFormatException,
            CommunicationException;

void  addSubject(String namespaceId, String subject) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            SubjectAlreadyExistsException,
            MessageFormatException,
            CommunicationException,
            OperationTimeoutException;

void  delSubject(String namespaceId, String subject) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            SubjectNotFoundException,
            CommunicationException,
            MessageFormatException,
            OperationTimeoutException;

String[]  listSubjects(String namespaceId) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            MessageFormatException,
            CommunicationException;

String[]  listSubjects(String namespaceId, String filter) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            MessageFormatException,
            CommunicationException;

void  addSubjectMapping (String  namespaceId,  int  mapType,
            String  subject,  String[]  subjectMapping,
            int resolveMode) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            SubjectNotFoundException,
            InvalidMappingException,
            MappingAlreadyExistsException,
            CommunicationException,
            MessageFormatException,
            OperationTimeoutException;

void  delSubjectMapping (String  namespaceId,  int  mapType,
            String subject) throws
            DatastoreAccessException,
            InvalidInputException,
            NamespaceNotFoundException,
            SubjectNotFoundException,
            MappingNotFoundException,
            CommunicationException,
            MessageFormatException,
            OperationTimeoutException;
```

```
String[]  listSubjectMappings (String  namespaceId,  int  mapType,
           String subject) throws
           DatastoreAccessException,
           InvalidInputException,
           NamespaceNotFoundException,
           SubjectNotFoundException,
           MappingNotFoundException,
           MessageFormatException,
           CommunicationException;

int  getResolveMode(String  namespaceId,  int  mapType,
           String subject) throws
           DatastoreAccessException,
           InvalidInputException,
           NamespaceNotFoundException,
           SubjectNotFoundException,
           MappingNotFoundException,
           MessageFormatException,
           CommunicationException;

void  setNotification(boolean state) throws
           MessageFormatException,
           CommunicationException;

}



public class NamespaceAdminFactory {

    public NamespaceAdminFactory(Properties myProp);

    public NamespaceAdminFactory(String myPropFile);
    public NamespaceAdmin create();
    public void delete(NamespaceAdmin admin);
}


/* Definition for Resolve Mode */
public class ResolveMode{
        public static final int ALGORITHMIC = 0;
        public static final int NON_ALGORITHMIC = 1;
        public static final int INVALID_VAL = -1;
};

/* Definition for Subject Mapping Mode */
public class MappingType {
        public static final int PUBLISH = 0;
        public static final int SUBSCRIBE = 1;
        public static final int INVALID_VAL = -1;
};
```

Properties Supported

The following property name-value pairs are supported for the NamespaceAdminFactory in Cisco Configuration Engine:

Property Name	Possible value	Applicable type provider
PROVIDER	Local/remote.	
STRATEGY	Either “secure” or “non_secure” to determine the type of context.	local
LOCK_TIMEOUT	The maximum waiting time for acquiring a file lock before throwing an exception.	Local
PROVIDER_URL	ldap://<ldap server ip>:<port>	local
SECURITY_PRINCIPAL	**dn** of the user object.	local
SECURITY_CREDENTIALS	password of the user object.	local
SECURITY_AUTHENTICATION	simple.	local
NAMESPACE_CONTEXT	**dn** of namespace context.	local
CONTEXT_POOL_SIZE	Integer that indicates the number of connections to maintain to the LDAP server.	local
RETRY_COUNT	Number of retry to acquire file lock.	local
TRANSPORT_ENCRYPTION	On/off	remote
SERVER_URL	URL of the server for the remove provider.	remote

Group Administration API Reference

C++ Version

There are some additional classes defined for the C++ administration API, CharString, CharStringArray, and GroupMemberArray. They are defined with a deep-copy constructor so that when the client application needs to copy one ReturnObject container into another, the appropriate deep-copy can be performed.

```
/*-------------------------------------------------------------------
 * GroupAdmin.h
 *
 * May 2004, Jane Jin
 *
 * Copyright (c) 2004 by Cisco Systems, Inc.
 * All rights reserved.
 *-------------------------------------------------------------------
 */

#ifndef _GROUPADMIN_H_
#define _GROUPADMIN_H_

#include "GroupMember.h"
#include "Properties.h"
#include "ReturnObjArray.h"
#include "ReturnStrArray.h"
#include "ErrorInfo.h"
#include "AdminConstants.h"


class GROUP_ADMIN_API GroupAdmin
{
  friend class GroupAdminFactory;

  public:

    enum {SUCCESS, FAILED};

    virtual int addMembers(
        const char* absGroupId,
        const GroupMember** members,
        const int memberCnt) = 0;

    virtual int deleteMembers(
        const char* absGroupId,
        const GroupMember** members,
        const int memberCnt) = 0;

    virtual int deleteAllMembers(
        const char* absGroupId) = 0;

    virtual int moveMembers(
        const char* srcAbsGroupId,
        const char* destAbsGroupId,
        const GroupMember** members,
        const int memberCnt) = 0;

    virtual int listGroups(
        const char* absGroupId,
        const int depth,
```

```
        const char* filter,
        ReturnStrArray& result) = 0;

    virtual int listMembers(
        const char* absGroupId,
        const char* filter,
        ReturnObjArray<GroupMember>& result) = 0;

    virtual int cloneMembers(
        const char* srcAbsGroupId,
        const char* destAbsGroupId,
        const GroupMember** members,
        const int memberCnt) = 0;

    virtual int cloneAllMembers(
        const char* srcAbsGroupId,
        const char* destAbsGroupId) = 0;

    virtual int cloneGroups(
        const char* srcAbsGroupId,
        const char* destAbsGroupId) = 0;

    virtual int renameGroup(
        const char* absGroupId,
        const char* newGroupId) = 0;

    virtual int isMember(
        const char* absGroupId,
        GroupMember* member,
        bool &ismember) = 0;

    virtual int listParents(
        const char* elementId,
        ReturnStrArray& result) = 0;

    virtual int setNotification(bool state) = 0;

    virtual const ErrorInfo* getErrorInfo() = 0;

  protected:
    virtual ~GroupAdmin(){};
};

#endif //_GROUPADMIN_H_
```

Java Version

```
package com.cisco.cns.admin.group;

public class GroupMember
{
public static final int GROUP = 0;
public static final int ELEMENT = 1;
public static final String[] Type = { "GROUP", "ELEMENT" };

public GroupMember(int type, String id);

public GroupMember(int type, String id, String absoluteId);

public String getId();
```

```
public String getAbsoluteId();

public int getType();

public String getTypeInString();

}

public class Group extends GroupMember
{
public Group(String id);

public Group(String id, String absoluteId);
}

public class Element extends GroupMember
{
public Element(String id);

public  Element(String id, String absoluteId)

}


/* GroupAdmin interface */
public interface GroupAdmin
{
    public static final int SCOPE_ALL = 0;
public static final int SCOPE_ONE = 1;

    void addMembers(String absGroupId, GroupMember[] members) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
ParentNotFoundException,
InvalidMemberException,
MemberAlreadyExistsException,
CommunicationException;

void deleteMembers(String absGroupId, GroupMember[] members) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
InvalidMemberException,
ParentNotFoundException,
MemberNotFoundException,
CommunicationException;

void deleteAllMembers(String absGroupId) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
ParentNotFoundException,
CommunicationException;

void moveMembers(String srcAbsGroupId,String destAbsGroupId,GroupMember[] members) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
InvalidMemberException,
```

```
InvalidOperationException,
ParentNotFoundException,
MemberNotFoundException,
MemberAlreadyExistsException,
CommunicationException;

GroupMember[] listMembers(String absGroupId, String filter) throws
DatastoreAccessException,
InvalidInputException,
ParentNotFoundException,
MessageFormatException,
CommunicationException;

String[] listGroups(String absGroupId, int depth, String filter) throws
DatastoreAccessException,
InvalidInputException,
ParentNotFoundException,
MessageFormatException,
CommunicationException;

void cloneMembers(String srcAbsGroupId, String destAbsGroupId, GroupMember[] members)
throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
InvalidMemberException,
ParentNotFoundException,
MemberNotFoundException,
MemberAlreadyExistsException,
CommunicationException;

void cloneAllMembers(String srcAbsGroupId, String destAbsGroupId) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
InvalidMemberException,
ParentNotFoundException,
MemberAlreadyExistsException,
CommunicationException;

void cloneGroups(String srcAbsGroupId, String destAbsGroupId) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
InvalidMemberException,
ParentNotFoundException,
MemberAlreadyExistsException,
CommunicationException;

void renameGroup(String absGroupId, String groupId) throws
DatastoreAccessException,
OperationTimeoutException,
MessageFormatException,
InvalidInputException,
MemberAlreadyExistsException,
ParentNotFoundException,
CommunicationException;

boolean isMember(String absGroupId, GroupMember member) throws
DatastoreAccessException,
InvalidInputException,
```

```
ParentNotFoundException,
MessageFormatException,
CommunicationException;

String[] listParents(String element) throws
DatastoreAccessException,
InvalidInputException,
MemberNotFoundException,
MessageFormatException,
CommunicationException;

void setNotification(boolean state) throws
MessageFormatException,
CommunicationException;
}


/* Factory class for GroupAdmin */
public class GroupAdminFactory
{
public:
GroupAdminFactory (Properties property);
GroupAdmin create ();
void delete (GroupAdmin ga);
};



package com.cisco.cns.admin.group;

public class MemberElementInfo
{

    public GroupMember getFailedElement()

    public GroupMember[] getCompletedElements()

    public GroupMember[] getRemainingElements()

}
```

Properties Supported

The following property name-value pairs table, originated from Unified Device Administrative Interface function specification, is supported for the GroupAdminFactory in Cisco Configuration Engine:

Property Name	Possible Values	Applicable Type Provider
PROVIDER	Class name of remote provider.	
TRANSPORT_ENCRYPTION	on/off	remote
SERVER_URL	URL of the server for the remote provider.	remote

Notification API Reference

C++ Version

```
// DataChangeInfo.h

enum MappingType
{
SUBSCRIBE,
PUBLISH
};


enum ResolveMode
{
NON_ALGORITHMIC,
ALGORITHMIC
};


class DataChangeInfo
{
friend class DataChangeNotification;

enum GROUP_OPERATION_TYPE {
                                    ADD_MEMBERS=0,
                                    DEL_MEMBERS,
                                    DEL_ALL_MEMBERS,
                                    MOVE_MEMBERS,
                                    CLONE_MEMBERS,
                                    CLONE_ALL_MEMBERS,
                                    CLONE_GROUPS,
                                    RENAME_GROUP,
                                    ENABLE_GROUP_NOTIFICATION,
                                    DISABLE_GROUP_NOTIFICATION
                                };

enum NAMESPACE_OPERATION_TYPE {
                                    ADD_NAMESPACE=100,
                                    DEL_NAMESPACE,
                                    CLONE_NAMESPACE,
                                    ADD_SUBJECT,
                                    DEL_SUBJECT,
                                    ADD_MAP,
                                    DEL_MAP,
                                    ENABLE_NAMESPACE_NOTIFICATION,
                                    DISABLE_NAMESPACE_NOTIFICATION
                                };

virtual  OperationType  getOpType();
};

class GroupChangeInfo : class DataChangeInfo
{
friend class DataChangeNotification;
friend class CallbackUtil;

public:

const char*  getSourceGroupId();
```

```
const char*  getDestinationGroupId();
void getMembers(ReturnObjArray<GroupMember> &members);
void getGroups(ReturnStrArray &groups);
void getElements(ReturnStrArray &elements);
~GroupChangeInfo();

};


class NamespaceChangeInfo : class DataChangeInfo
{
friend class DataChangeNotification;
    friend class CallbackUtil;

public:

const char* getNamespaceId();
const char* getSubject();
MappingType  getMapType();
void getMapping(ReturnStrArray &mapping);
ResolveMode  getResolveMode();
const char* getCloneId();
~NamespaceChangeInfo();

};


// DataChangeNotification.h

class NotificationCallback
{
public:
virtual  void  onNotification ( DataChangeInfo *dataChangeInfo,
               char*  name_space,
               void*  callbackArg  )=0;
};

class DataChangeNotification
{
public:

virtual int startNotification () = 0;  /* Notification Registration */
virtual int stopNotification () = 0;  /* Notification Un-registration */

};

class GroupChangeNotification : public DataChangeNotification
{
public:
     GroupChangeNotification (const char*  name_space,
                     NotificationCallback *callback,
                void* callbackArg);
     virtual int startNotification ();  /* Notification Registration */

     virtual int stopNotification ();  /* Notification Un-registration */

     virtual ~GroupChangeNotification();
};

class NamespaceChangeNotification : public DataChangeNotification
{
public:
     NamespaceChangeNotification (const char*  name_space,
                     NotificationCallback *callback,
```

```
                void* callbackArg);
     virtual int startNotification ();  /* Notification Registration */

     virtual int stopNotification ();  /* Notification Un-registration */

     virtual ~NamespaceChangeNotification();
};
```

Java Version

```
package com.cisco.cns.notification;

public class DataChangeInfo
{
   public static final int ADD_MEMBERS = 0;
   public static final int DEL_MEMBERS = 1;
   public static final int DEL_ALL_MEMBERS = 2;
   public static final int MOVE_MEMBERS = 3;
   public static final int CLONE_MEMBERS = 4;
   public static final int CLONE_ALL_MEMBERS = 5;
   public static final int RENAME_GROUP = 6;
   public static final int ENABLE_GROUP_NOTIFICATION = 7;
   public static final int DISABLE_GROUP_NOTIFICATION = 8;

   public static final int ADD_NAMESPACE = 100;
   public static final int DEL_NAMESPACE = 101;
   public static final int CLONE_NAMESPACE = 102;
   public static final int ADD_SUBJECT = 103;
   public static final int DEL_SUBJECT = 104;
   public static final int ADD_MAP = 105;
   public static final int DEL_MAP = 106;
   public static final int ENABLE_NAMESPACE_NOTIFICATION = 107;
   public static final int DISABLE_NAMESPACE_NOTIFICATION = 108;

   public int getOpType ()
};

public class GroupChangeInfo extends DataChangeInfo

public String getSourceGroupId()

public String getDestinationGroupId()

public GroupMember[] getMembers()

public String[] getGroups()

public String[] getElements()
};

public class NamespaceChangeInfo extends DataChangeInfo
{

    public String getNamespaceId ()

    public String getSubject ()

    public int getMapType ()

    public int getResolveMode ()
```

```
    public String getCloneId ()

    public String[] getMapping ()

};


public interface NotificationHandler
{
    void  onNotification ( DataChangeInfo dataChangeInfo,
                           String namespace,
                           Object callbackArg ) throws
                           NotificationHandlerException;
};



public class GroupChangeNotification
{
    public GroupChangeNotification (String namespace,
                                    NotificationHandler callback,
                                    Object callbackArg);


    public int startNotification () throws NotificationException;

    public int stopNotification () throws NotificationException;

};

public class NamespaceChangeNotification
{

    public NamespaceChangeNotification (String namespace,
                                    NotificationHandler callback,
                                    Object callbackArg);

    public int startNotification () throws NotificationException;

    public int stopNotification () throws NotificationException;

};
```

CHAPTER 7

Device Administration Interface API

This chapter presents the software functional specification for a device administrative interface that enables you to programmatically create, delete, modify and query agent-enabled and non-agent-enabled devices.

Setup

Solaris Java Environment

The CLASSPATH needs to contain the following, where PKG_DIR is set to */<INSTALLDIR>/CSCOesdk*:

```
$PKG_DIR/conf
$PKG_DIR/templates/java/rules
$PKG_DIR/java/GroupAdmin.jar
$PKG_DIR/java/NSMAdminClient.jar
$PKG_DIR/java/DevAdminClient.jar
$PKG_DIR/java/cnsadminutils.jar
$PKG_DIR/java/log4j.jar
$PKG_DIR/java/JceBlowfish.jar
$PKG_DIR/lib/tibrvj.jar
$PKG_DIR/java/velocity-1.4.patch.jar
$PKG_DIR/java/commons-collections.jar
$PKG_DIR/java/commons-httpclient-2.0.jar
$PKG_DIR/java/commons-logging.jar
$PKG_DIR/java/commons-digester.jar
$PKG_DIR/java/commons-beanutils-1.6.1.jar
$PKG_DIR/java/xerces.jar
```

The sample program can be references in **$PKG_DIR/sample/cns_admin/java** for java.

Solaris C++ environment

Step 1 Add */<INSTALLDIR>/CSCOesdk/lib* to the LD_LIBRARY_PATH environment variable.

Step 2 Add */<INSTALLDIR>/CSCOesdk/bin* to the PATH environment variable.

Step 3 Set the following:

```
setenv XML_TEMPLATE_PATH /<INSTALLDIR>/CSCOesdk/templates/cpp
setenv CNS_ADMIN_PROP_PATH /<INSTALLDIR>/CSCOesdk/conf
setenv ADMIN_LOG_PROP_PATH /<INSTALLDIR>/CSCOesdk/conf
```

Step 4 The libraries to be linked with from the $PKG_DIR/sample/cns_admin/cpp/Makefile are:

ADMIN_LIBS=-lHttpWrapper -lACE -ludiclient -lGroupAdmin -lnsmadmin -lxerces-c2_5 -lcnsadminutils

The sample program can be reference in $PKG_DIR/sample/cns_admin/cpp for C++.

Software Architecture

The Device Administration Interface consists of a set of APIs that your application can invoke. The interface provides local and remote implementations, for in-process and out-process applications respectively.

An example of an in-process application is the Cisco Configuration Engine GUI, and an example of an out-process application is a management application running external to Cisco Configuration Engine.

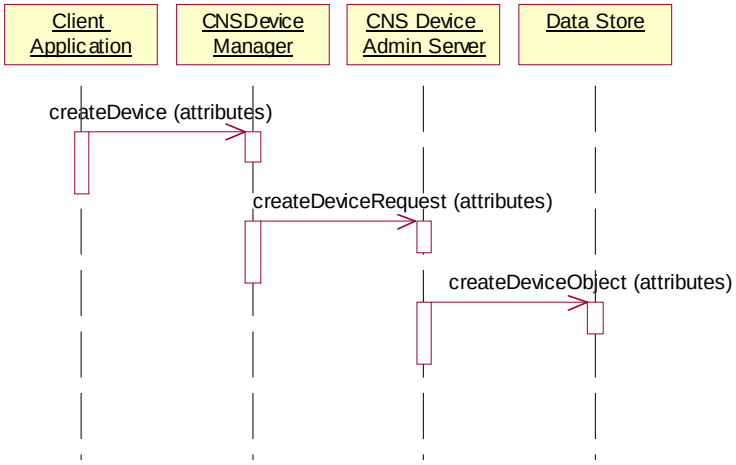
This release of the SDK supports only the remote implementation. The remote implementation accomplishes tasks through a Device Administration Server in Cisco Configuration Engine, by communicating with it over HTTP transport. The Administration Server, in turn modifies the data store as specified in the request from the API.

Sequence of Operations

Remote Implementation

Figure 7-1 shows the sequence of operations for a create device operation for a remote implementation of the client API.

Figure 7-1 Sequence of Operations for Remote Client API



1. The client application makes a call to the CNSDeviceManager which is a part of the SDK.
2. The device manager object communicates the request to the Device Administration Server running in Cisco Configuration Engine over HTTP transport.
3. The server then modifies the data store to service the request and responds with the status of the data store operation.

End User Interface

This section describes the methods in the Administrative interface.

The Device Administration Interface has been designed based on the following principles:

- All attributes of the device common to all services are administered by the class **CNSDeviceManager**.
- All attributes specific to a service are administered by classes derived from **DeviceServiceAttributes**.
- The device name is used as the key to look up device objects. Therefore, this string must be unique to every device in the data store.

Status of Operations

The method of indicating the status of each operation is through error codes in the C++ implementation. When a client written in C++ receives an error code, it can examine an **DeviceErrorInfo** object that can be obtained from the *getDeviceErrorInfo* method. This object contains information about the precise nature of the error.

In the Java implementation, an exception is thrown for each type of error.

For the function signatures, please refer to Java Version of Device Interface API, page 7-41 or C++ Version of Device Interface API, page 7-32.

Thread Safety

All methods in the C++ and Java implementation are thread-safe.

Class Transport

This class represents the transport through which the device can be reached. This is an abstract class that has methods common to all types of transport.

Get Transport Type

This method returns the transport type of the transport. Possible values are Transport::CNS_AGENT_TRANSPORT and Transport::TELNET_TRANSPORT.

Signature

TransportType getTransportType()

Return Values

Integer that gives the transport type.

Get Transport Identifier

This method returns the unique identifier for the transport. A device could potentially have multiple transports, i.e. one path through the COMM server and another through a direct Telnet. This identifier identifies each transport uniquely.

Note The current schema does not support this, but this may be supported in future and the admin interface must be able to handle this situation.

Signature

String getTransportId()

Return Values

String that identifies the transport.

Class CNSAgentTransport

This class extends off the Transport class and represents the CNS Agent transport mechanism.

Constructor

The constructor creates an instance of the CNSAgentTransprt object.

Signature

CNSAgentTransport(String transportId)

Argument

transportId – String that uniquely identifies the transport instance for the device.

Class AgentProxyTransport

This class extends off the Transport class and represents a Telnet transport mechanism. It provides methods for you to add hop information to reach the device.

Constructor

The constructor creates an instance of the AgentProxyTransprt object.

Signature

AgentProxyTransport(String transportId, String gatewayId, String transportCategory)

Arguments

transportId – String that uniquely identifies the transport instance for the device.

gatewayId – String that gives the ID of the gateway that handles operations for the device.

transportCategory – String that gives the category of the non-agent transport to the device.

Set Gateway ID

This method sets the gateway ID for the device.

Signature

void setGatewayId (String gatewayId)

Arguments

gatewayId – String that gives the ID of the gateway that handles operations for the device.

Return Values

None.

Get Gateway ID

This method gets the gateway ID for the device.

Signature

String getGatewayId ()

Return Values

String that gives the gateway ID.

Set Hop Info

This method sets the hop information for the device.

Signature

void setHopInfo (HopInfo[] info)

Arguments

info – An array of HopInfo objects that give the hop information for the device.

Return Values

None.

Get Hop Info

This method gets the hop information for the device.

Signature

HopInfo[] getHopInfo ()

Return Values

An array of HopInfo objects that give the hop information for the device.

Class HopInfo

This class represents the hop information for a device. It refers to a single hop.

The number of hops is transport-specific. Typically, multiple hops are required to reach the device through Telnet.

All of the fields in the HopInfo class are optional as not all hop types require all the fields. In the event that one of them is irrelevant to the hop type, strings can be substituted with “” and integers with 0.

Note This object is passed to the CNSDeviceManager through the Transport object and setting hop info attributes on the local object does not modify the device object in the data store.

Constructor

The constructor constructs a HopInfo object.

Signature

HopInfo (String hopType, String ipAddr, int port, String userName, String password)

Arguments

hopType – String that indicates the type of hop. The type of hop can be one of the pre-defined types in Cisco Configuration Engine, or user-defined.

ipAddr – String that gives the IP address for the hop.

port – Integer that indicates the port to which to Telnet. This applies to the case when the hop is a COMM Server hop.

userName – String that gives the username to login as.

password – String that gives the password for userName.

Set Hop Type

Method to set the hop type string.

Signature

void setHopType (String hopType)

Arguments

hopType - String that indicates the type of hop. The type of hop can be one of the pre-defined types in Cisco Configuration Engine, or user-defined.

Set IP Address

Method to set the IP address of the hop.

Signature

void setIpAddr(String ipAddr)

Arguments

ipAddr – String that gives the IP address for the hop.

Set Port

Method to set the port number for the hop.

Signature

void setPort (int port)

Arguments

port - Integer that indicates the port to which to Telnet. This applies to the case when the hop is a COMM Server hop.

Set Username

Method to set the username for the hop

Signature

void setUserName(String userName)

Arguments

userName – String that gives the username to login as.

Set Password

Method to set the password for the hop.

Signature

void setPassword (String password)

Arguments

password – String that gives the password for the username used for logging

Get Hop Type

Method that returns the hop type of the hop.

Signature

String getHopType ()

Return Values

String that gives the hop type

Get IP Address

Method that returns the IP address of the hop.

Signature

String getIPAddr ()

Return Values

String that gives the IP address of the hop

Get Port

Method that returns the port number of the hop.

Signature

int getPort ()

Return Values

Integer that gives the port number of the hop

Get Username

Method that returns the username of the hop.

Signature

String getUserName ()

Return Values

String that gives the username of the hop

Get Password

Method that returns the password of the hop.

Signature

String getPassword ()

Return Values

String that gives the password of the hop.

Class AgentProxyConfiguration

This class provides methods to add, delete and query error and ignore patterns for the scripts used to communicate to devices through telnet. This object can be created through the DeviceAdminFactory.

Add Error Pattern

Method to add error pattern.

Signature

int addErrorPattern(String deviceCategory, String pattern)

Arguments

deviceCategory – device category

pattern – String that contains the error pattern

Return Value

Integer that gives the status of the operation.

Delete Error Pattern

Method to delete error pattern.

Signature

int deleteErrorPattern(String deviceCategory, String pattern)

Arguments

deviceCategory – device category

pattern – String that contains the error pattern

Return Value

Integer that gives the status of the operation.

Get Error Pattern

Method to get all error patterns.

Signature

String[] getErrorPattern(String deviceCategory)

Arguments

deviceCategory – device category

Return Value

String array of error patterns

Add Ignore Pattern

Method to add ignore pattern.

Signature

int addIgnorePattern(String deviceCategory, String pattern)

Arguments

deviceCategory – device category

pattern – String that contains the ignore pattern

Return Value

Integer that gives the status of the operation.

Delete Ignore Pattern

Method to delete ignore pattern.

Signature

int deleteIgnorePattern(String deviceCategory, String pattern)

Arguments

deviceCategory – device category

pattern – String that contains the ignore pattern

Return Value

Integer that gives the status of the operation.

Get Ignore Pattern

Method to get all ignore patterns.

Signature

String[] getIgnorePattern(String deviceCategory)

Arguments

deviceCategory – device category

Return Value

String array of ignore patterns

Class DeviceServiceAttribute

This class represents the service specific attributes of a device.

Get Service Type

This method returns the type of service to which the device attributes apply.

Signature

ServiceType getServiceType ()

Return Value

A ServiceType value (integer) which can be CONFIGURATION, EXEC or IMAGE_DISTRIBUTION.

Register Service

Method to register the device with a service.

Signature

int registerService (String deviceName, String id, Transport transport)

Arguments

deviceName – Name of the device.

ID – String that gives the unique identifier for the device in the service domain.

> **Example:** If the object is of type ConfigurationAttributes, this ID refers to the config ID of the device. If the object is of type ImageServiceAttributes, this refers to the image ID of the device.

transport – The transport mechanism to communicate with the device for this service.

Return Value

Integer that gives the status of the operation.

Unregister Service

Method to unregister the device from a service.

Signature

int unregisterService (String deviceName)

Arguments

deviceName – Name of the device.

Return Value

Integer that gives the status of the operation.

Set Service Transport

Method to set the transport mechanism to be used for this service.

Signature

int setServiceTransport (String deviceName, Transport transport)

Arguments

deviceName – Name of the device.

transport – The transport mechanism to communicate with the device for this service.

Return Value

Integer that gives the status of the operation.

Get Service Transport

Method to get the transport mechanism for this service.

Signature

int getServiceTransport (String deviceName)

Arguments

deviceName – Name of the device.

Return Value

Transport object containing the transport attributes for this service.

Set Device Identifier

Method to add set the device identifier.

Signature

int setId(String deviceName, String id)

Arguments

deviceName – Name of the device.

id – String that gives the unique identifier for the device.

Return Value

Integer that gives the status of the operation.

Get Device Identifier

Method to add get the device identifier.

Signature

String getId(String deviceName)

Arguments

deviceName – Name of the device.

Return Value

String that gives the event identifier of the device.

Set Device Password

Method to add set the device authentication password for PIX devices.

Signature

int setPassword(String deviceName, String password)

Arguments

deviceName – Name of the device.

password – String that gives the password for the device.

Return Value

Integer that gives the status of the operation.

Get Device Password

Method to add get the PIX device password.

Signature

String getPassword(String deviceName)

Arguments

deviceName – Name of the device.

Return Value

String that gives the password of the device.

Class ConfigurationAttributes

This class derives from DeviceServiceAttribute and represents the Configuration Service attributes of the device.

Set Template

This method sets the template attribute on the device.

Signature

int setTemplateNames (String configId, String []templateNames)

Arguments

configId – String that refers to config Id of the device.

templateNames – String array that gives the names of the template files to be used for the device.

Return Values

Integer that gives the result of the operation.

Add Template

This method adds a template on the device.

Signature

int addTemplateName (String configId, String templateName)

Arguments

configId – String that refers to config Id of the device.

templateName – String that gives the name of the template file to be added for the device.

Return Values

Integer that gives the result of the operation.

Get Template

This method gets the templates associated with the device.

Signature

String[] getTemplateNames (String configId)

Arguments

configId – String that refers to config Id of the device.

Return Values

String array that gives the template file(s) associated with the device.

Delete Template

This method that deletes templates associated with the device.

Signature

int delTemplateNames (String configId, String[] templateNames)

Arguments

configId – String that refers to config Id of the device.

templateNames – String array that gives the templates to be deleted from the device.

Return Values

Integer that gives the status of the operation.

Class CNSDevice

This class encapsulates the common attributes for all CNS devices.

Constructor

The constructor creates an object of type CNSDevice.

Signature

CNSDevice (String deviceName, String identifier)

Arguments

deviceName – String that gives the name of the linecard device.

identifier – the event id of the device.

Get Device Type

This method returns the type of device.

Signature

int getDeviceType()

Return Value

Integer that gives the type of device. Possible values are CNSDevice.PIX_DEVICE, CNSDevice.GENERIC_DEVICE, and CNSDevice.LINE_CARD.

Get Device Identifier

This method returns the identifier of the device.

Signature

String getId ()

Return Value

String that gives the identifier of the device.

Get Device Name

This method returns the nameof the device.

Signature

String getName ()

Return Value

String that gives the name of the device.

Class PIXDevice

This class derives from CNSDevice and represents a PIX device.

Constructor

The constructor creates an object of type PIXDevice.

Signature

PIXDevice (String deviceName, String identifier, String password, String configAction, String errorAction)

Set Password

This method allows you to set the PIXDevice password.

Signature

void setPassword (String password)

Arguments

Password – String that gives the password for the PIX device.

Get Password

Method that returns the password of the PIX device.

Signature

String getPassword ()

Return Value

String that gives the device password.

Set Configuration Action

This method allows you to set the PIXDevice configuration action.

Signature

void setConfigAction (String configAction)

Arguments

configAction – String that specifies the PIX devices configuration action.

Get Configuration Action

Method that returns the configuration action of the PIX device.

Signature

String getConfigAction ()

Return Value

String that gives the device configuration action.

Set Error Action

This method allows you to set the PIXDevice error action.

Signature

void setErrorAction (String errorAction)

Arguments

errorAction – String that specifies the PIX devices error action.

Get Error Action

Method that returns the error action of the PIX device.

Signature

String getErrorAction ()

Return Value

String that gives the device error action.

Class ASADevice

This class derives from PIXDevice and represents a ASA device.

Constructor

The constructor creates an object of type ASADevice.

Signature

ASADevice (String deviceName, String identifier, String password, String configAction, String errorAction)

Set Password

This method allows you to set the ASADevice password.

Signature

void setPassword (String password)

Arguments

Password – String that gives the password for the ASA device.

Get Password

Method that returns the password of the ASA device.

Signature

String getPassword ()

Return Value

String that gives the device password.

Set Configuration Action

This method allows you to set the ASADevice configuration action.

Signature

void setConfigAction (String configAction)

Arguments

configAction – String that specifies the ASA devices configuration action.

Get Configuration Action

Method that returns the configuration action of the ASA device.

Signature

String getConfigAction ()

Return Value

String that gives the device configuration action.

Set Error Action

This method allows you to set the ASADevice error action.

Signature

void setErrorAction (String errorAction)

Arguments

errorAction – String that specifies the ASA devices error action.

Get Error Action

Method that returns the error action of the ASA device.

Signature

String getErrorAction ()

Return Value

String that gives the device error action.

Class LineCardDevice

This class represents a module of a device. The module can be managed in the same way as a device and is hence classified as a type of device.

Constructor

Signature

LineCardDevice(String deviceName, String identifier, String lineCardType)

Arguments

deviceName – String that gives the name of the linecard device.

identifier – the event id of the device.

linecardType – String that gives the line card type.

Class DeviceAdminFactory

This factory class creates and destroys instances of CNSDeviceManager and DeviceServiceAttr objects. The objects are created with the properties specified in the constructor.

Constructor

This method creates a DeviceAdminFactory object.

Signature

DeviceAdminFactory (Properties properties)

Arguments

properties – A Property object that indicates the transport, security, provider parameters that the created objects should use.

Create CNSDeviceManager Object

This method creates and returns a CNSDeviceManager object.

Signature

CNSDeviceManager createDeviceManager ()

Return Values

A CNSDeviceManager with the properties and device type specified.

Delete CNSDeviceManager Object

This method destroys an instance of the CNSDeviceManager object.

Signature

void deleteDeviceManager (CNSDeviceManager mgr)

Arguments

mgr – The instance of CNSDeviceManager that must be destroyed.

Create LineCardManager Object

This method creates and returns a LineCardDeviceManager object with the set of properties and device type that are passed to the Factory object.

Signature

LineCardManager createLineCardManager ()

Return Values

A LineCardManager with the properties specified in the constructor of the factory object.

Delete LineCardManager Object

This method destroys an instance of the LineCardManager object.

Signature

void deleteLineCardManager (LineCardManager mgr)

Arguments

mgr – The instance of LineCardManager that must be destroyed.

Create DeviceServiceAttr Object

This method creates and returns a DeviceServiceAttr object.

Signature

DeviceServiceAttribute createServiceObject (ServiceType serviceType)

Arguments

serviceType – Argument that gives the type of service object to be constructed. Possible values include DeviceAdminFactory.

SERVICE_TYPE_CONFIG, DeviceAdminFactory. SERVICE_TYPE_IMAGE.

Return Values

A DeviceServiceAttribute with the properties and device type specified.

Delete DeviceServiceAttr Object

This method destroys an instance of the DeviceServiceAttribute object.

Signature

void deleteServiceObject (DeviceServiceAttribute serviceObject)

Arguments

serviceObject – The instance of DeviceServiceAttribute that must be destroyed.

Create AgentProxyConfiguration Object

This method creates and returns a AgentProxyConfiguration object.

Signature

AgentProxyConfiguration createTransportConfigObject()

Return Values

An AgentProxyConfiguration object.

Delete AgentProxyConfiguration Object

This method destroys an instance of the AgentProxyConfiguration object.

Signature

void deleteTransportConfigObject(AgentProxyConfiguration agentProxyConfig)

Arguments

agentProxyConfig – The instance of AgentProxyConfiguration that must be destroyed.

Class CNSDeviceManager

This class encapsulates the attributes and methods required to create device objects and set device parameters that are common to all services.

CreateDevice

Method to create a device object with a given name and identifier. This identifier string is assigned to the EventId, ConfigId. By default, the device is added to the *default* group.

Signature

Status createDevice (CNSDevice device)

Arguments

device – Device object whose attributes give the attributes of the device to be created.

Return Value

Integer indicating status.

CreateDevice

Method to create a device object with a given name and identifier. This identifier string is assigned to the EventId, ConfigId. In addition, the device is added as a member of the mentioned groups.

Signature

Status createDevice (CNSDevice device, String[] groupIds)

Arguments

device – Device object whose attributes give the attributes of the device to be created.

Return Value

Integer indicating status.

CreateDevice

Method to create a device object with a given name and identifier. This identifier string will be assigned to the Event Id, Config Id. The attrs[] can be used selectively to assign value to attributes.

Signature

Status createDevice (CNSDevice device, CNSAttribute attrs[])

Arguments

device – Device object whose attributes give the attributes of the device to be created.

attrs – array of attribute objects.

Return Value

Integer indicating status.

CreateDevice

Method to create a device object with a given name and identifier. This identifier string will be assigned to the Event Id, Config Id. The attrs[] can be used selectively to assign value to attributes. In addition, the device will be added as a member of the mentioned groups.

Signature

Status createDevice (CNSDevice device, CNSAttribute attrs[], String[] groupIds)

Arguments

device – Device object whose attributes give the attributes of the device to be created.

attrs – array of attribute objects.

groupIds – string array of group Ids.

Return Value

Integer indicating status.

Rename Device

Method to set device name.

Signature

Status renameDevice(String currentName, String newName)

Arguments

currentName – String that denotes the current name of the device.

newName – String that denotes the name to replace the current name.

Return Value

Integer indicating status.

Get Device Type

Method that fetches the device type.

Signature

String getDeviceType(String deviceName)

Arguments

deviceName - String that indicates the device name.

Return Value

String that indicates the device type.

Set Event ID

Method to add set the event ID attribute on the device object.

Signature

Status setDeviceId(String deviceName, String id)

Arguments

deviceName – String that denotes the name of the device.

ID – String that gives the event ID of the device.

Return Value

Integer that gives the status of the operation.

Get Event ID

Method to add get the event ID attribute of the device object.

Signature

String getDeviceId(String deviceName)

Arguments

deviceName – String that denotes the name of the device.

Return Value

String that gives the event ID of the device.

Set Device Attributes

Method to add set an attribute value on the device. The schema of the data store must support the attribute being set. One or more values may be assigned to it. Only string values are supported.

Signature

Status setDeviceAttributes(String deviceName, CNSAttribute[] attrs)

Arguments

deviceName – String that denotes the name of the device.

attrs – A set of CNSAttribute objects.

Return Value

Integer that gives the status of the operation.

Get Device Attribute

Method to get an attribute value from the device.

Signature

String[] getDeviceAttribute(String deviceName, String attrName)

Arguments

deviceName – String that denotes the name of the device.

attrName – String that gives the attribute name.

Return Value

An array containing the values of the device attribute.

Delete Device Attribute

Method to delete an attribute value from the device.

Signature

int deleteDeviceAttribute(String deviceName, String attrName)

Arguments

deviceName – String that denotes the name of the device.

attrName – String that gives the attribute name.

Return Value

Integer indicating the status of the operation.

Delete Device Object

Method to delete a device.

Signature

Status deleteDevice (String deviceName)

Arguments

deviceName – String that denotes the name of the device.

Return Value

Integer that gives the status of the operation.

List Device Objects

Method to list all device names known to the Cisco-CE instance. If the device type argument is specified, only list devices of that type. If the device type argument is CNSDeviceManager.ALL_DEVICES, list all device names known to the Cisco-CE instance.

Signature

String[] listDevices(int deviceType)

Arguments

deviceType – Integer that gives the device type of the devices that must be listed. Possible values are CNSDevice.PIX_DEVICE, CNSDevice.ASA_DEVICE, CNSDevice.GENERIC_DEVICE , CNSDevice.LINE_CARD, and CNSDevicemanager.ALL_DEVICES.

Return Value

An array of strings containing the list of device names.

List Device Objects Based on Condition

Method to list all device names known to the Cisco Configuration Engine instance based on an attribute value.

- If the device type argument is specified, only list device objects of that type.
- If the device type argument is CNSDevice.ALL_DEVICES, list all device objects that satisfy the condition.

The returned object will contain the attribute name-value pairs for the attributes requested in the method call.

Signature

ResultObject[] listDevices(String condition, String[] attrs)

Arguments

condition – String of the form “attrName==attrValue”, where

attrName and attrValue are strings denoting a device.

attribute name and value respectively.

attrs – The attributes to return as part of the result.

Return Value

An array of ResultObject objects.

List All Device Attribute Names

Method to list all the attribute names of the given device type. This method will be useful to supply the attribute names required for the listDevices method discussed in List Device Objects Based on Condition.

Signature

String[] listAllAttributeNames(int deviceType)

Arguments

deviceType – Integer that gives the device type of the devices that must be listed. Possible values are CNSDevice.PIX_DEVICE, CNSDevice.ASA_DEVICE, CNSDevice.GENERIC_DEVICE, and CNSDevice.LINE_CARD.

Return Value

An array of Strings that give all the attribute names of the device object.

Get All Registered Services

Method that returns the set of services that with which the device is registered. The set of services with which a device is registered is determined by the number of services that have knowledge of the device.

Note In our implementation, creation of a device automatically registers itself with the configuration service. Whereas, the device needs to be explicitly registered with the Image Service.

So this call will always return DeviceServiceAttribute.CONFIGURATION and DeviceServiceAttribute.EXEC. Whether this device is registered with the image service, or not, is determined by the presence of this device with an image ID in the image server space.

If an object with the given device name exists with an image ID in the image device container, DeviceServiceAttribute.IMAGE_DISTRIBUTION will also be returned.

Signature

ServiceType[] getRegisteredServices(String deviceName)

Arguments

deviceName – Name of the device.

Return

An array of integers representing the service types that the device is registered for.

Class LineCardManager

This class encapsulates methods specific to line card administration.

Associate Subdevice

Method to associate a subdevice object with a given main device name.

Signature

Status associateSubDevice (String mainDeviceName, String subDeviceName)

Arguments

mainDeviceName – String that gives the name of the main device.

subDeviceName – String that gives the name of the subdevice.

Return Value

Integer indicating status.

Disassociate Subdevice

Method to disassociate a subdevice object with a given main device name.

Signature

Status disassociateSubDevice (String mainDeviceName, String subDeviceName)

Arguments

mainDeviceName – String that gives the name of the main device.

subDeviceName – String that gives the name of the subdevice.

Return Value

Integer indicating status.

List Subdevices

Method to list all subdevice names for a given main device.

Signature

String[] listSubDeviceNames(String deviceName)

Arguments

deviceName – String that denotes the name of the device.

Return Value

An array of strings containing the list of device names.

Get Line Card Type

Method that fetches the linecard type if the object is a subdevice.

Signature

String getLineCardType(String deviceName)

Arguments

deviceName - String that indicates the device name.

Return Value

String that indicates the line card type.

Get Parent Device

Method that returns the parent device name for a given subdevice. If there is no parent device associate, this method returns a null.

Signature

String getParentDevice (String subDeviceName)

Arguments

subDeviceName – String that gives the name of the subdevice.

Return Value

String that gives the main device name for the sub device.

Class ResultAttribute

This class represents a device attribute and its values.

Get Attribute Name

This method sets the name of the attribute.

Signature

String getName ()

Return Values

String that gives the name of the device.

Get Attribute Values

This method returns the values for the attribute.

Signature

Object[] getValues ()

Return Value

Object array that gives the set of values for the device.

Class ResultObject

This class represents an object returned as part of a search or list operation. It has a name and a set of ResultAttribute objects.

Get Attributes

This method returns the set of attributes of the device requested as part of the search operation.

Signature

ResultAttribute[] getAttributes ()

Return Value

Array of ResultAttribute objects.

Get Name

This method returns the name of the ResultObject.

Signature

String getName ()

Return Value

String that gives the name of the object.

Class ResultSetIterator

This class is an iterator that can be used to iterate through the result set returned by a search operation.

Get Next Object

This method returns the object next to the one the iterator is pointing to and moves the next reference to the object being returned. If the end of the result set is reached, this method returns NULL.

Signature

ResultObject next()

Return Value

ResultObject object that is next in the result set.

Get Next *n* Objects

This method returns the next *n* objects from the one the iterator is pointing to and moves the "next" reference to the last object being returned, where *n* is an integer passed as an argument. If the number of objects remaining is less than *n*, it returns as many objects as it can. If the end of the result set is reached, it returns a NULL.

Signature

ResultObject[] next(int count)

Arguments

count – Integer that gives the number of objects to return.

Return Value

An array of ResultObject objects.

Configuration and Restrictions

A property file is required for the client components. The property file specifies the implementation to use, the transport to use if applicable, the URL of the Administration Server, if applicable and security settings if any.

For the local implementation of the client, more properties specify data store specific parameters.

The following table lists the property names and possible values.

Property Name	Possible Values	Type Provider
PROVIDER	local/remote	
TRANSPORT_ENCRYPTION	on/off	remote
SERVER_URL	URL of the server for the remote provider	remote

Security

The Device Administration interface provides transport encryption for the remote implementation. However, authentication and authorization are outside the scope of the interface and the calling application must enforce your credentials before invoking the methods of this interface, if security is desired.

The interface is extensible so that future enhancements for passing in authentication/authorization tokens can be made.

To address the requirement of existing users who use the Configuration Server with directory credentials, the local implementation of the interface provides a way to specify the directory username and password. These credentials are used to access the directory. If no credentials are specified, all operations throw an InvalidCredentials exception.

All applications other than Configuration Server are expected to use the no-security implementation. In this implementation, the credentials supplied by you are used. If no credentials are specified, a set of credentials specified at setup time are used.

C++ Version of Device Interface API

API Definition

```
typedef int TransportType;

class UDIAPI Transport
{
    friend class CNSAgentTransport;
    friend class AgentProxyTransport;

public:
    enum {CNS_AGENT_TRANSPORT, AGENT_PROXY_TRANSPORT};

    virtual TransportType getTransportType() const;
    virtual const char *getTransportId() const;
    virtual ~Transport();

protected:
    void setTransportType (TransportType type);
    void setTransportId(const char *id);

private:
    TransportType _transportType;
    char *_transportId;
    Transport();
};


class UDIAPI CNSAgentTransport : public Transport
{
public:
    CNSAgentTransport (const char *transportId);
    virtual ~CNSAgentTransport ();

private:
};
```

```
class UDIAPI AgentProxyTransport : public Transport
{
public:
    AgentProxyTransport (const char *transportId, const char *gatewayId,
                                 const char *transportCategory);
    void setGatewayId (const char *gatewayId);
    const char *getGatewayId ();

    void setCategory(const char *category);
    const char *getCategory ();

    void setHopInfo (HopInfo *info, int hopCount);
    void getHopInfo (ReturnObjArray<HopInfo> &hopInfo);

protected:
    virtual ~AgentProxyTransport();

private:
    char *_gatewayId;
    char *_category;
    HopInfo *_hopInfo;
    int _hopCount;
};


class ErrorInfo
{
public:
    ErrorInfo (int statusCode, const char *description);
    ErrorInfo (int statusCode, const char *description, boolean transient);
    ErrorInfo (int statusCode, const char *description, boolean transient, const char
                                            **ids);
    int getStatusCode();
    const char *getDescription();
    boolean isTransient();
private:
    int _statusCode;
    ACE_CString description;
    boolean transient;
};


class UDIAPI DeviceServiceAttribute
{
public:
    virtual ~DeviceServiceAttribute();

    enum { CONFIG, EXEC, IMAGE };

    virtual int registerService (
        const char *deviceName,
        const char *id,
        Transport &transport)=0;

    virtual int unregisterService (
        const char *deviceName)=0;

    virtual int setServiceTransport (
        const char *deviceName,
        Transport &transport)=0;

    virtual int getServiceTransport (
        const char *deviceName,
```

```
        ReturnObjArray<Transport> &transport)=0;

    virtual int getServiceType ();

    virtual int setId(
        const char *deviceName,
        const char *identifier)=0;

    virtual int getId(
        const char *deviceName,
        ReturnStrArray &id)=0;

    virtual int setPassword (
        const char *deviceName,
        const char *password)=0;

    virtual int getPassword(
        const char *deviceName,
        ReturnStrArray &password)=0;

    virtual const DeviceErrorInfo *getErrorInfo()=0;

    virtual void setErrorInfo(int code, const char *desc, bool transient);

    virtual void setErrorInfo(DeviceErrorInfo *errorInfo);

protected:
    int _serviceType;

    DeviceServiceAttribute();

};


class UDIAPI ConfigurationAttributes : public DeviceServiceAttribute
{
public:
    ConfigurationAttributes();
    virtual ~ ConfigurationAttributes();

    virtual int registerService (
        const char *deviceName,
        const char *id,
        Transport &transport);

    virtual int unregisterService (
    const char *deviceName);

    virtual int setServiceTransport (
        const char *deviceName,
        Transport &transport);

    virtual int getServiceTransport (
        const char *deviceName,
        ReturnObjArray<Transport> &transport);

    virtual int getServiceType ();

    virtual int setId(
        const char *deviceName,
        const char *identifier);

    virtual int getId(
        const char *deviceName,
```

```
        ReturnStrArray &id);

    virtual int setPassword (
        const char *deviceName,
        const char *password);

    virtual int getPassword(
        const char *deviceName,
        ReturnStrArray &password);

    virtual int setTemplateNames (
        const char *configId,
        const char **templateNames,
        int count);

    virtual int addTemplateNames (
        const char *configId,
        const char **templateNames,
        int count);

    virtual int getTemplateNames (
        const char *configId,
        ReturnStrArray &templateNames);

    virtual int delTemplateNames (
        const char *configId,
        const char **templateNames,
        int count);

    virtual const DeviceErrorInfo *getErrorInfo();

    virtual void setErrorInfo(int code, const char *desc, bool transient);

    virtual void setErrorInfo(DeviceErrorInfo *errorInfo);

protected:

};


class UDIAPI ImageServiceAttributes : public DeviceServiceAttribute
{
public:
    ImageServiceAttributes();
    virtual ~ ImageServiceAttributes();

    virtual int registerService (
        const char *deviceName,
        const char *id,
        Transport &transport);

    virtual int unregisterService (
        const char *deviceName);

    virtual int setServiceTransport (
        const char *deviceName,
        Transport &transport);

    virtual int getServiceTransport (
        const char *deviceName,
        ReturnObjArray<Transport> &transport);

    virtual int getServiceType ();
```

```
    virtual int setId(
        const char *deviceName,
        const char *identifier);

    virtual int getId(
        const char *deviceName,
        ReturnStrArray &id);

    virtual int setPassword (
        const char *deviceName,
        const char *password);

    virtual int getPassword(
        const char *deviceName,
        ReturnStrArray &password);

    virtual const DeviceErrorInfo *getErrorInfo();

    virtual void setErrorInfo(int code, const char *desc, bool transient);

    virtual void setErrorInfo(DeviceErrorInfo *errorInfo);
};


class UDIAPI CNSDevice
{
public:
    enum {PIX_DEVICE, GENERIC_DEVICE, LINE_CARD};

    CNSDevice(const char *deviceName, const char *identifier);
    CNSDevice();
    virtual ~CNSDevice();
    int getDeviceType();
    const char *getName();
    const char *getId();

protected:
    char *_name;
    char *_id;
    int _deviceType;
};


class UDIAPI PIXDevice : public CNSDevice
{
public:
    PIXDevice (const char *deviceName, const char *identifier, const char *password);
    virtual ~PIXDevice();
    void setPassword (const char *password);
    const char *getPassword();

private:
    char *_password;
};


class UDIAPI LineCardDevice : public CNSDevice
{
public:
    LineCardDevice (const char *deviceName, const char *identifier, const char
                        *lineCardType);
    virtual ~LineCardDevice();
    void setLineCardType (const char *lineCardType);
    const char *getLineCardType();
```

```
private:
    char *_lineCardType;
};


class UDIAPI CNSDeviceManager
{
public:
    enum TransportType {CNS_AGENT_TRANSPORT, AGENT_PROXY_TRANSPORT};
    enum DeviceType {PIX_DEVICE, GENERIC_DEVICE, LINE_CARD};
    enum {ALL_DEVICES=1000};
    enum {SUCCESS, FAILED};

    virtual int createDevice (CNSDevice &device)=0;
    virtual int createDevice (CNSDevice &device, CNSStringAttribute attrs[],
                              int attr_count)=0;
    virtual int createDevice (CNSDevice &device, const char **groupIds, int
                              group_count)=0;
    virtual int createDevice (CNSDevice &device, CNSStringAttribute attrs[],
                              int attr_count, const char **groupIds, int group_count)=0;
    virtual int renameDevice (const char *currentDeviceName, const char *newDeviceName)=0;
    virtual int getDeviceType(const char *deviceName, int &deviceType)=0;
    virtual int setEventId (const char *deviceName, const char *id)=0;
    virtual int getEventId (const char *deviceName, ReturnStrArray &id)=0;
    virtual int setDeviceAttributes(const char *deviceName,
                      CNSStringAttribute attrs[],
                                                int attrCount)=0;
    virtual int getDeviceAttribute(const char *deviceName, const char *attrName,
    ReturnStrArray &retStrArray)=0;
    virtual int delDeviceAttribute (const char *deviceName, const char *attrName)=0;
    virtual int deleteDevice (const char * deviceNames)=0;
    virtual int listDevices(int deviceType, ReturnStrArray &deviceNames)=0;
    virtual int listDevices(int deviceType, const char *condition, const char ** attrs,
                      int attrCount,
                                      ReturnObjArray<ResultObject> &result)=0;
    virtual int listAllAttributeNames(int deviceType, ReturnStrArray &attrNames)=0;
    virtual int getRegisteredServices(const char *deviceName,
                      ReturnObjArray<int> &retAgents)=0;

    virtual const DeviceErrorInfo *getErrorInfo()=0;

protected:
    virtual void setErrorInfo(int code, const char *desc, bool transient)=0;
};
class UDIAPI LineCardManager
{
public:
    virtual int associateSubDevice (const char *mainDeviceName, const char
*subDeviceName)=0;
    virtual int disassociateSubDevice (const char *mainDeviceName, const char
*subDeviceName)=0;
    virtual int getParentDevice (const char *subDeviceName,
                      ReturnStrArray &mainDeviceName)=0;
    virtual int getLineCardType(const char *deviceName, ReturnStrArray &linecardType)=0;
    virtual int listSubDeviceNames (const char *mainDeviceName,
                      ReturnStrArray &subdeviceNames)=0;
    virtual const DeviceErrorInfo *getErrorInfo()=0;

protected:
    virtual void setErrorInfo(int code, const char *desc, bool transient)=0;
};
```

```
class UDIAPI AgentProxyConfiguration
{
public:
    virtual int addErrorPattern(const char * deviceCategory, const char * pattern) = 0;
    virtual int deleteErrorPattern(const char * deviceCategory, const char * pattern) = 0;
    virtual int getErrorPattern(const char * deviceCategory, ReturnStrArray &patterns)=0;
    virtual int addIgnorePattern(const char * deviceCategory, const char * pattern) = 0;
    virtual int deleteIgnorePattern(const char * deviceCategory, const char * pattern) =
                            0;
    virtual int getIgnorePattern(const char * deviceCategory, ReturnStrArray &patterns) =
                            0;
    virtual const DeviceErrorInfo *getErrorInfo()=0;

protected:
    virtual void setErrorInfo(int code, const char *desc, bool transient)=0;

    virtual void setErrorInfo(DeviceErrorInfo *errorInfo)=0;

};



class UDIAPI CNSStringAttribute
{
public:
    const char *getName() const;
    const char **getValues (int &count) const;

    CNSStringAttribute();
    ~CNSStringAttribute();
    CNSStringAttribute(const char *name);
    CNSStringAttribute(const char *name, const char **values, int count);

    CNSStringAttribute& operator=(const CNSStringAttribute& attr);
    CNSStringAttribute(const CNSStringAttribute& attr);

    void setName(const char *name);

    void setValues (const char **values, int count);
    void addValue (const char *value);

private:
    void deleteValues();

    char *_name;
    char **_values;
    int _value_count;
};


template <class T>
class UDIAPI CNSAttribute
{
public:
    CNSAttribute();
    ~CNSAttribute();
    CNSAttribute& operator=(const CNSAttribute& attr);
    CNSAttribute(const CNSAttribute& attr);
    CNSAttribute(const char *name);
    CNSAttribute(const char *name, const T *values, int count);

    const char *getName() const;
    void setName(const char *name);
```

```
    const T *getValues(int &count) const;
    void setValues(const T *values, int count);
    void addValue(T &value);

private:
    void deleteValues();

    char *_name;
    T *_values;
    int _value_count;
};


typedef CNSStringAttribute ResultAttribute;


class UDIAPI ResultObject
{
    friend class DeviceParser;
public:
    const ResultAttribute **getAttributes (int &count) const;
    const char *getName() const;
    ResultObject(const ResultObject& resObj);

private:
    ResultObject();
    ResultObject(const char *name);
    void setName(const char *name);
    void setAttributes(const char *name, const ResultAttribute **attributes, int count);

    ResultAttribute **_attrs;
    char *_name;
    int _attrs_count;
};


class ResultSetIterator
{
public:
    ResultSetIterator (ResultObject [], int numObjects);
    ResultObject next();
    ResultObject **next(int count);

private:
    ResultObject **_result;
    int _result_count;
};


class UDIAPI HopInfo
{
public:
    HopInfo();
    ~HopInfo();
    HopInfo& operator=(const HopInfo& hopInfo);
    HopInfo(const char *hopType, const char *ipAddr, int port,
                   const char *user, const char *passwd);
    void setHopType(const char *hopType);
    void setIPAddr(const char *ipAddr);
    void setPort(int port);
    void setUserName(const char *user);
    void setPassword(const char *passwd);

    const char *getHopType() const;
```

```
    const char *getIPAddr() const;
    int getPort() const;
    const char *getUserName() const;
    const char *getPassword() const;

private:
    char *m_hopType;
    char *m_ipAddr;
    int m_port;
    char *m_user;
    char *m_passwd;
};


class UDIAPI DeviceErrorInfo
{
public:
    DeviceErrorInfo (int statusCode, const char *description);
    DeviceErrorInfo (int statusCode, const char *description, bool transient);
    DeviceErrorInfo (int statusCode, const char *description, bool transient,
                          const char **ids);
    virtual ~DeviceErrorInfo();
    int getStatusCode() const;
    const char *getDescription() const;
    bool isTransient() const;

private:
    int _statusCode;
    char *description;
    bool transient;
};
```

Return Codes

```
0 - Success
-1 - Failure
Error Codes
#define INVALID_INPUT_PARAMETER 201
#define FATAL_SERVER_COMMUNICATION 202
#define TEMPORARY_SERVER_COMMUNICATION 203
#define OBJECT_ALREADY_EXISTS 204
#define CREATE_OBJECT_DATA_STORE_ERROR 205
#define DATA_STORE_BUSY 206
#define ADD_DEVICE_TO_GROUP_ERROR 207
#define INVALID_CARD_TYPE 208
#define INVALID_MAIN_DEVICE 209
#define NO_SUCH_OBJECT 210
#define ID_NOT_UNIQUE 211
#define DATA_STORE_SCHEMA_ERROR 212
#define TEMPLATE_NAME_MISMATCH 213
#define OPERATION_TIMEOUT_ERROR 214
#define OPERATION_FAILED_ERROR 215
#define NETWORK_ERROR 216
#define NOT_SUPPORTED 217
#define INTERNAL_ERROR 218
```

Java Version of Device Interface API

API Signature

package com.cisco.cns.admin.device;

```
public class DeviceAdminFactory
{
    public static int SERVICE_TYPE_CONFIG = 0;
    public static int SERVICE_TYPE_EXEC = 1;
    public static int SERVICE_TYPE_IMAGE = 2;
    public DeviceAdminFactory(Properties properties);
    public DeviceAdminFactory(String propFile);
    public CNSDeviceManager createDeviceManager();
    public void deleteDeviceManager(CNSDeviceManager dev_mgr);
    public LineCardManager createLineCardManager();
    public void deleteLineCardManager(LineCardManager line_card_mgr);
    public DeviceServiceAttribute createServiceObject(int serviceType);
    public void deleteServiceObject(DeviceServiceAttribute serviceObj);
    public AgentProxyConfiguration createTransportConfigObject();
    public void deleteTransportConfigObject(AgentProxyConfiguration agentProxyConfig);
};

public class Transport
{
    public static final int CNS_AGENT_TRANSPORT = 0;
    public static final int AGENT_PROXY_TRANSPORT = 1;
    public int getTransportType() ;
    public void setTransportId(String id);
    public String getTransportId();
    protected int _transportType;
    protected String _transport_id;
};

public class CNSAgentTransport extends Transport
{
    public CNSAgentTransport (String transportId);
};

public class AgentProxyTransport extends Transport
{
    AgentProxyTransport (String transportId, String gatewayId, String transportCategory);
    public void setGatewayId (String gatewayId);
    public String getGatewayId ();
    public void setCategory (String transportCategory);
    public String getCategory ();
    public void setHopInfo (HopInfo[] info);
    public HopInfo[] getHopInfo ();
};

public abstract class DeviceServiceAttribute
{
    public static final int CONFIGURATION = 0;
    public static final int EXEC = 1;
    public static final int IMAGE_DISTRIBUTION = 2;
    public int getServiceType ();
    public int setId(String deviceName, String id) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
```

```
                OperationTimedOutException;

public String getId(String deviceName) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException;

public int setPassword(String deviceName, String password) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
                OperationTimedOutException;

public String getPassword(String deviceName) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException;

public int registerService (String deviceName, String id, Transport transport) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
                OperationTimedOutException;

public int unregisterService (String deviceName) throws
                InvalidParameterException,
NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
                OperationTimedOutException;

public int setServiceTransport(String deviceName, Transport transport) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
                OperationTimedOutException;

public Transport getServiceTransport(String deviceName) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException;
};

public class ConfigurationAttributes extends DeviceServiceAttribute
{
public int setTemplateNames (String configId, String[] templates) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                ObjectNotFoundException,
                OperationTimedOutException;

public int addTemplateNames (String configId, String[] templates) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException,
                OperationTimedOutException;
```

```
public String[] getTemplateNames (String configId) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException;

public int delTemplateNames (String configId, String[] templates) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException;
};

public class CNSDevice
{
    public static final int PIX_DEVICE = 0;
    public static final int ASA_DEVICE = 4;
    public static final int GENERIC_DEVICE = 1;
    public static final int LINE_CARD = 2;
    public CNSDevice (String deviceName, String identifier);
    public int getDeviceType ();
    public String getName();
    public String getId();
    protected String _deviceName;
    protected String _id;
    protected int deviceType;
};

public class PIXDevice extends CNSDevice
{
    public PIXDevice (String deviceName, String identifier, String password, String
    configAction, String errorAction);
    public void setPassword (String password);
    public String getPassword();
    public void setConfigAction(String configAction);
    public String getConfigAction();
    public void setErrorAction (String errorAction);
    public String getErrorAction();
    private String _password;
    private String configAction;
    private String errorAction;
}
public class ASADevice extends CNSDevice
{
    public ASADevice (String deviceName, String identifier, String password, String
    configAction, String errorAction);
    public void setPassword (String password);
    public String getPassword();
    public void setConfigAction(String configAction);
    public String getConfigAction();
    public void setErrorAction (String errorAction);
    public String getErrorAction();
    private String _password;
    private String configAction;
    private String errorAction;
}

public class LineCardDevice extends CNSDevice
{
    public LineCardDevice (String deviceName, String identifier, String lineCardType);
    public void setLineCardType (String password);
    public String getLineCardType();
```

```
    private String _lineCardType;
}
public class CNSDeviceManager
{
    public static final int ALL_DEVICES = 1000;
    // return code
    public static final int SUCCESS = 0;
    public static final int FAILED = 1;

public int createDevice (CNSDevice device)
               throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectAlreadyExistsException,
               OperationTimedOutException;

public int createDevice (CNSDevice device, String[] groupIds) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectAlreadyExistsException,
               OperationTimedOutException

public int createDevice (CNSDevice device, CNSAttribute attrs[]) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectAlreadyExistsException,
               OperationTimedOutException;

public int createDevice (CNSDevice device, CNSAttribute attrs[],
        String[] groupIds) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectAlreadyExistsException,
               OperationTimedOutException;

public int renameDevice (String currentDeviceName, String newDeviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException

public int getDeviceType (String currentDeviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException

public int setEventId (String deviceName, String identifier) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException

public String getEventId (String deviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException
```

```
public int setDeviceAttributes (String deviceName, CNSAttribute[] attrs) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException

public Object getDeviceAttribute(String deviceName, String attrName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException

public int delDeviceAttribute(String deviceName, String attrName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException

public int deleteDevice (String deviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               OperationTimedOutException;

public String[] listDevices(int deviceType) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException;

public ResultObject[] listDevices(String condition, String[] attrs, int deviceType)
               throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException;

public String[] listAllAttributeNames( int deviceType) throws
               NetworkException,
               OperationFailedException;

public int[] getRegisteredServices (String deviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException;
};

public interface LineCardManager {
    public int associateSubDevice (String mainDeviceName, String subDeviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException

public int disassociateSubDevice (String mainDeviceName, String subDeviceName)
               throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException,
               OperationTimedOutException
```

```
public String getParentDevice (String subDeviceName)
               throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException

public String getLineCardType (String currentDeviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException

public String[] listSubDeviceNames(String mainDeviceName) throws
               InvalidParameterException,
               NetworkException,
               OperationFailedException,
               ObjectNotFoundException
};

public class CNSAttribute
{
    public String getName ();
    public Object []getValues ();
    public CNSAttribute ();
    public CNSAttribute (String name);
    public CNSAttribute (String name, Object []values);
    public void setValues (Object []values);
    public void addValue (Object value);
};

public class ResultAttribute
{
    public String getName ();
    public Object[] getValues ();
    public ResultAttribute();
    public ResultAttribute(String name);
    public ResultAttribute(String name, Object []values);
    public void setValues (Object []values);
    public void addValue (Object value);
};

public class ResultObject
{
    public ResultAttribute[] getAttributes ();
    public String getName ();
    ResultObject();
    ResultObject(String name);
    ResultObject(String name);
    void setName(String name);
    void setAttributes(String name, ResultAttribute[] attributes);
};

public class ResultSetIterator
{
    public ResultObject next();
    public ResultObject[] next(int count);
};

public interface AgentProxyConfiguration
{
    public int addErrorPattern(String deviceCategory, String pattern) throws
               InvalidParameterException,
```

```
                NetworkException,
                OperationFailedException;

    public int deleteErrorPattern(String deviceCategory, String pattern) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException;

    public String[] getErrorPattern(String deviceCategory) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException;

    public int deleteIgnorePattern(String deviceCategory, String pattern) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException;

    public String[] getIgnorePattern(String deviceCategory) throws
                InvalidParameterException,
                NetworkException,
                OperationFailedException;
};

public class HopInfo
{
    public HopInfo();
    public HopInfo(String hopType, String ipAddr, int port, String user, String passwd)
        throws InvalidParameterException

    public void setHopType(String hopType)
        throws InvalidParameterException

    public void setIPAddr(String ipAddr)
        throws InvalidParameterException

    public void setPort(int port)
        throws InvalidParameterException

    public void setUserName(String user)
        throws InvalidParameterException

    public void setPassword(String passwd)
        throws InvalidParameterException;

    public String getHopType();
    public String getIPAddr();
    public int getPort();
    public String getUserName();
    public String getPassword();
};
```

Exceptions

```
/* This exception occurs when the client application passes invalid parameters */
public class InvalidParameterException
{
    InvalidParameterException (int statusCode, String description);
    public String getDescription();
    public int getStatusCode();
    private int _statusCode;
    private String _description;
```

```
};

/* This exception occurs when the remote implementation of the client API
encounters an error while communicating with the Admin Server*/
public class NetworkException
{
    NetworkException (int statusCode, String description);
    public String getDescription();
    public int getStatusCode();

    private int _statusCode;
    private String _description;
};

/* This exception occurs when an error occurs while processing the client request due to
data store problems or user error*/
public class OperationFailedException
{
    OperationFailedException (int statusCode, String description, bool transient);
    public String getDescription();
    public int getStatusCode();
    public bool isTransient(); // true if error is temporary

    private int _statusCode;
    private String _description;
    private bool _transient();
};

/* This exception occurs when the client application tries to create an object that
already exists*/
class ObjectAlreadyExistsException
{
    ObjectAlreadyExistsException (int statusCode, String description, String[] ids);
    public String getDescription();
    public int getStatusCode();
    public String[] getIdsWithError(); //Get device ids that caused the error

    private int _statusCode;
    private String _description;
    private String[] _ids;
};

/* This exception occurs when the client application tries to delete an object that does
not exist*/
class ObjectNotFoundException
{
    ObjectNotFoundException (int statusCode, String description, String[] ids);
    public String getDescription();
    public int getStatusCode();
    public String[] getIdsWithError(); //Get device ids that caused the error

    private int _statusCode;
    private String _description;
    private String[] _ids;
};
class OperationTimedOutException
{
    OperationTimedOutException (int statusCode, String description, String[] ids);
    public String getDescription();
    public int getStatusCode();
    public String[] getIdsWithError(); //Get device ids that caused the error

    private int _statusCode;
    private String _description;
```

```
    private String[] _ids;
};
```

CHAPTER 8

Creating Provisioning Solution

The Cisco Configuration Engine SDK provides the building blocks for writing provisioning applications. The SDK makes use of the following components on the Cisco Configuration Engine:

- Configuration Server—defines and activates configuration templates. Delivers configuration commands to the IOS device.
- Event Gateway/IMGW — enables network elements to publish and subscribe to events.
- Directory Server — data storage device based on Lightweight Directory Access Protocol (LDAP) version 3 compliant directories (Internal/External).
- Namespace Mapper (NSM) Service—provides a lookup service for managing logical groups of devices based on application, device/group ID, and event (Optional).

Creating Provisioning Solutions

There are three ways for the Cisco Configuration Engine SDK to interact with an IOS device:

- Pull operation
- Push operation
- Two-stage commit

These operations are described in this chapter.

Partial Configuration Using Cisco Configuration Engine

Partial Configuration of the device enables an operator or a provisioning application to incrementally update device configuration through Cisco Configuration Engine.

Partial Configuration happens through the mechanism of Publish and Subscribe. The device subscribes to a configuration event, and listens on a pre-determined subject on the Event Bus. The provisioning application, when it needs to configure a particular device or group of devices, publishes on this subject. The published configuration event is received by the listening device(s), which act on it by applying the incremental configuration.

For the publish-subscribe mechanism to work, the Event Agent and the Configuration Agent must be running on the device. The Event Agent enables event-based communication, and the Configuration Agent subscribes to configuration events. These agents can be turned on as part of the Initial Configuration process.

Configuration ID and Event ID

Every device, in addition to a Configuration ID, also has a unique Event ID. The Configuration ID is used when the device pulls it configuration from Cisco Configuration Engine. The Event ID is used when the device participates in Event-Based communication. The Configuration ID will be part of the bootstrap configuration, while the Event ID can be set through the Initial Configuration Template.

If the Event ID is not set, the Event ID assumes the value of the device hostname. For this reason, either the hostname or the Event ID must be configured before the device begins event-based communication. The Configuration ID and Event ID could be the same for a given device. But, no two devices may share a Configuration ID, and no two devices may share an Event ID.

Applications that publish configuration events can use the Cisco Configuration Engine SDK to publish events on the Event Bus.

When the Event Agent and the Configuration Agent are enabled on the device, and a unique Configuration ID and Event ID are available, the device is ready for partial updates.

Pull and Push Modes

Two modes are available to users for partial updates, pull and push. In the Pull mode, the configuration template for the device is imported to Cisco Configuration Engine. An event is then sent to the device that triggers the device to pull its configuration from Cisco Configuration Engine.

In the Push mode, the CLI commands are part of the payload of the event. In this mode, the template is not stored in Cisco Configuration Engine.

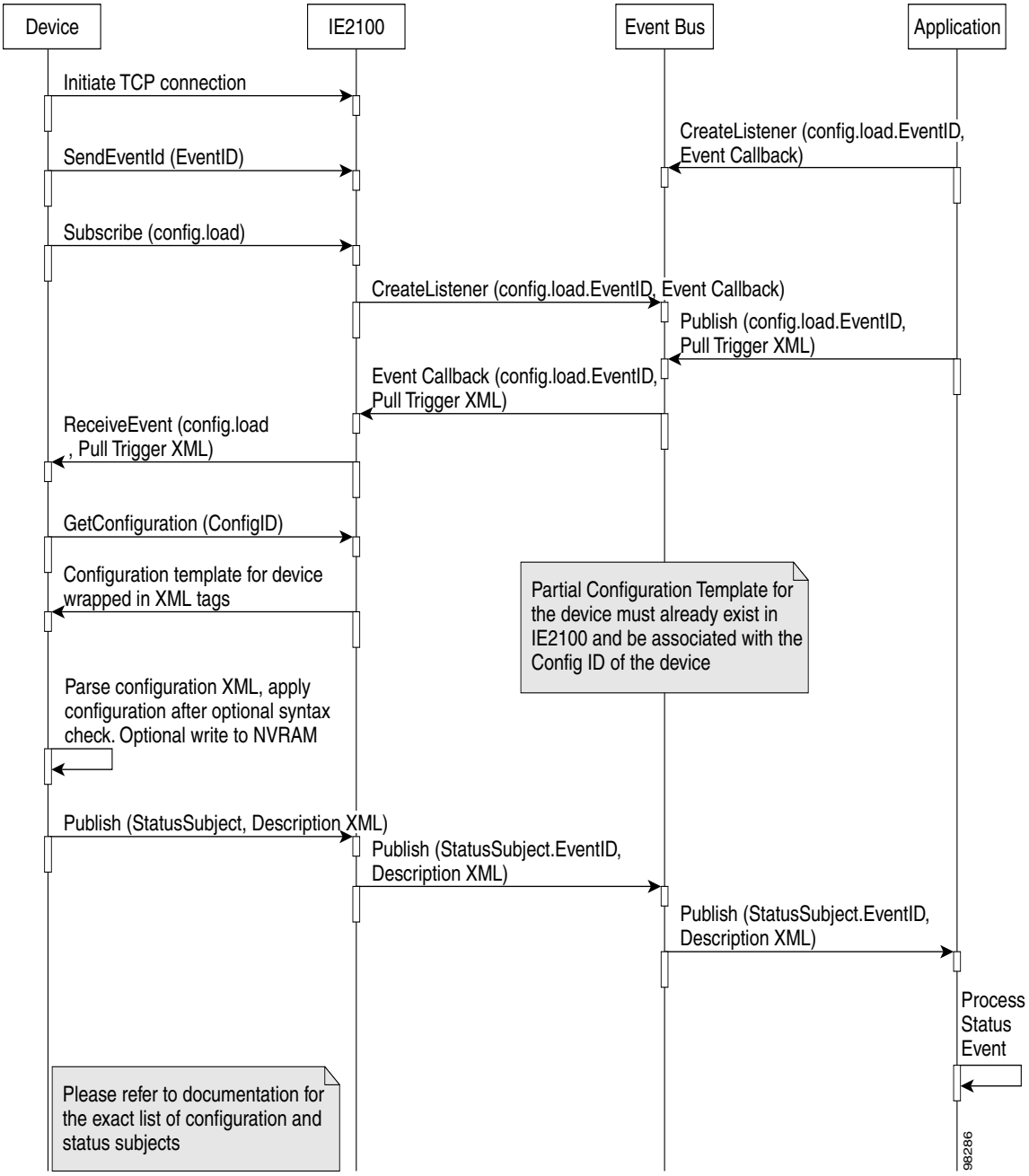
Sequence of Operations in Pull Mode

A pre-requisite for Pull mode is that a configuration template must exist for the device that is being triggered, and that template must be associated with the Configuration ID of the device in question.

- When the event agent begins execution on the device, it initiates a TCP connection to Cisco Configuration Engine, and passes its Event ID, therefore identifying itself by that ID for event-based communication.
- When the configuration agent on the device begins execution, it starts subscribing to a configuration subject (cisco.cns.config.load).
- This subscription request is received by the Event Gateway component of Cisco Configuration Engine, and it creates a listener for the configuration subject, further qualified by the Event ID of that device. It registers an event callback that will be executed when an event is published on the listening subject.
- When an external application has a configuration update to be sent to the device, it starts subscribing to the status subject from the device, qualified by the Event ID of the device. If this step is not done, the application will lose any status events published from the device.
- It then publishes an event on the configuration subject, with the payload of the event indicating that it is a pull event.
- The Event Callback registered by Cisco Configuration Engine for the configuration subject will be executed.
- The payload of the event will be passed to the device along with the subject.
- The device, upon receiving this trigger, will then contact Cisco Configuration Engine for its configuration template, passing in its Configuration ID.

- The Config Server component of Cisco Configuration Engine looks up the template for the device's Configuration ID, wraps the configuration in XML and sends it to the device.
- The device parses the XML, applies the configuration and publishes a status message, which will be available on the Event Bus through Cisco Configuration Engine.
- Since the external application created a listener for the status subject, the application's event callback will be called, where the status message can be processed.

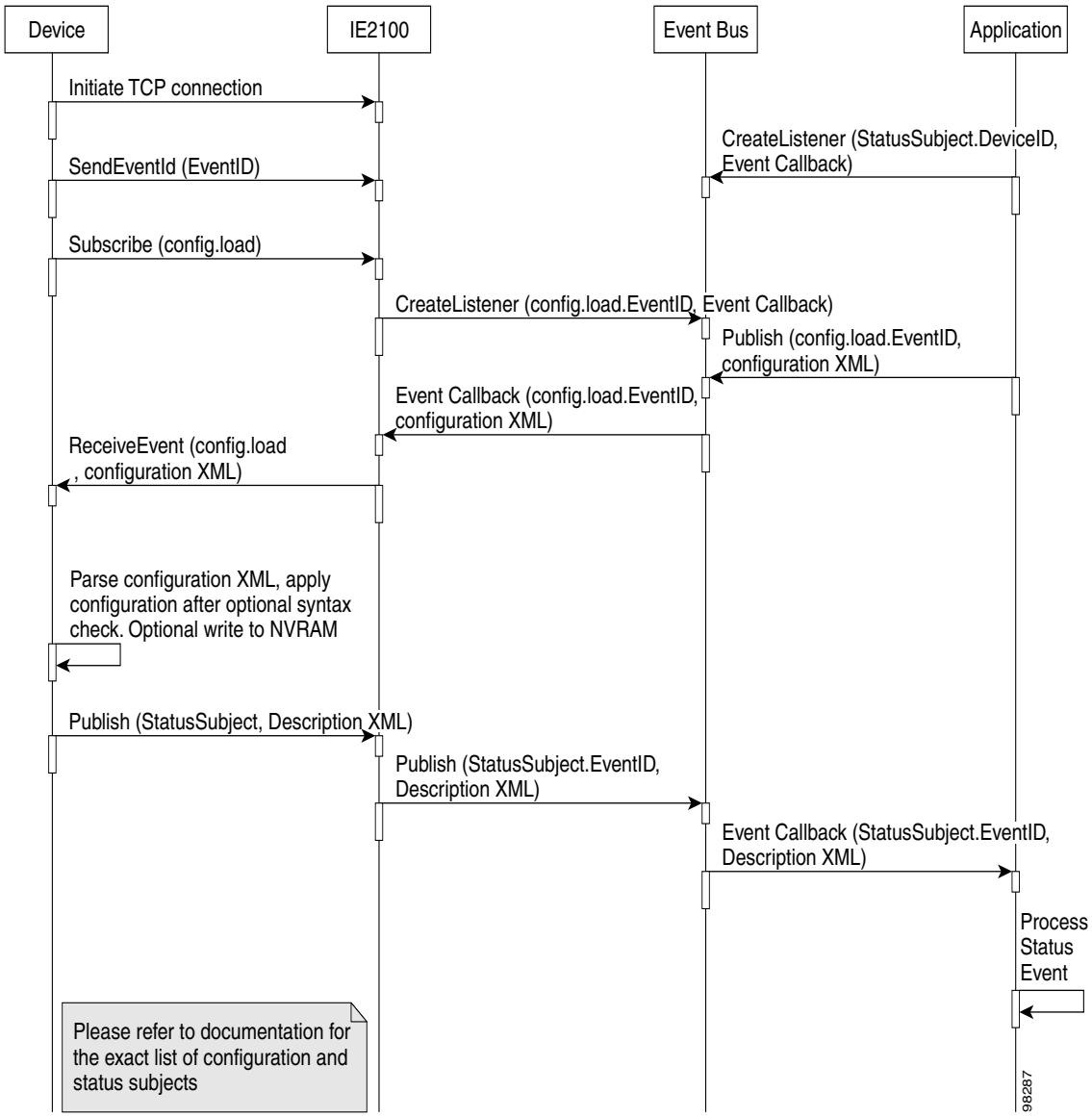
Figure 8-1 Sequence Diagram for Pull Mode



Sequence of Operations in Push Mode

- When the event agent begins execution on the device, it initiates a TCP connection to Cisco Configuration Engine, and passes its Event ID, therefore identifying itself by that ID for event-based communication.
- When the configuration agent on the device begins execution, it starts subscribing to a configuration subject (cisco.cns.config.load).
- This subscription request is received by the Event Gateway component of Cisco Configuration Engine, and it creates a listener for the configuration subject, further qualified by the Event ID of that device. It registers an event callback that will be executed when an event is published on the listening subject.
- When an external application has a configuration update to be sent to the device, it starts subscribing to the status subject from the device, qualified by the Event ID of the device. If this step is not done, the application will lose any status events published from the device.
- It then publishes an event on the configuration subject, with the payload of the event indicating that it is a push event, with the actual CLI commands embedded in the payload.
- The Event Callback registered by Cisco Configuration Engine for the configuration subject will be executed.
- The payload of the event will be passed to the device along with the subject.
- The device, upon receiving the event, parses the XML payload, applies the configuration CLI commands, and publishes a status message, which will be available on the Event Bus through Cisco Configuration Engine.
- Since the external application created a listener for the status subject, the application's event callback will be called, where the status message can be processed.

Figure 8-2 Sequence Diagram for Push Model



Two-stage Commit

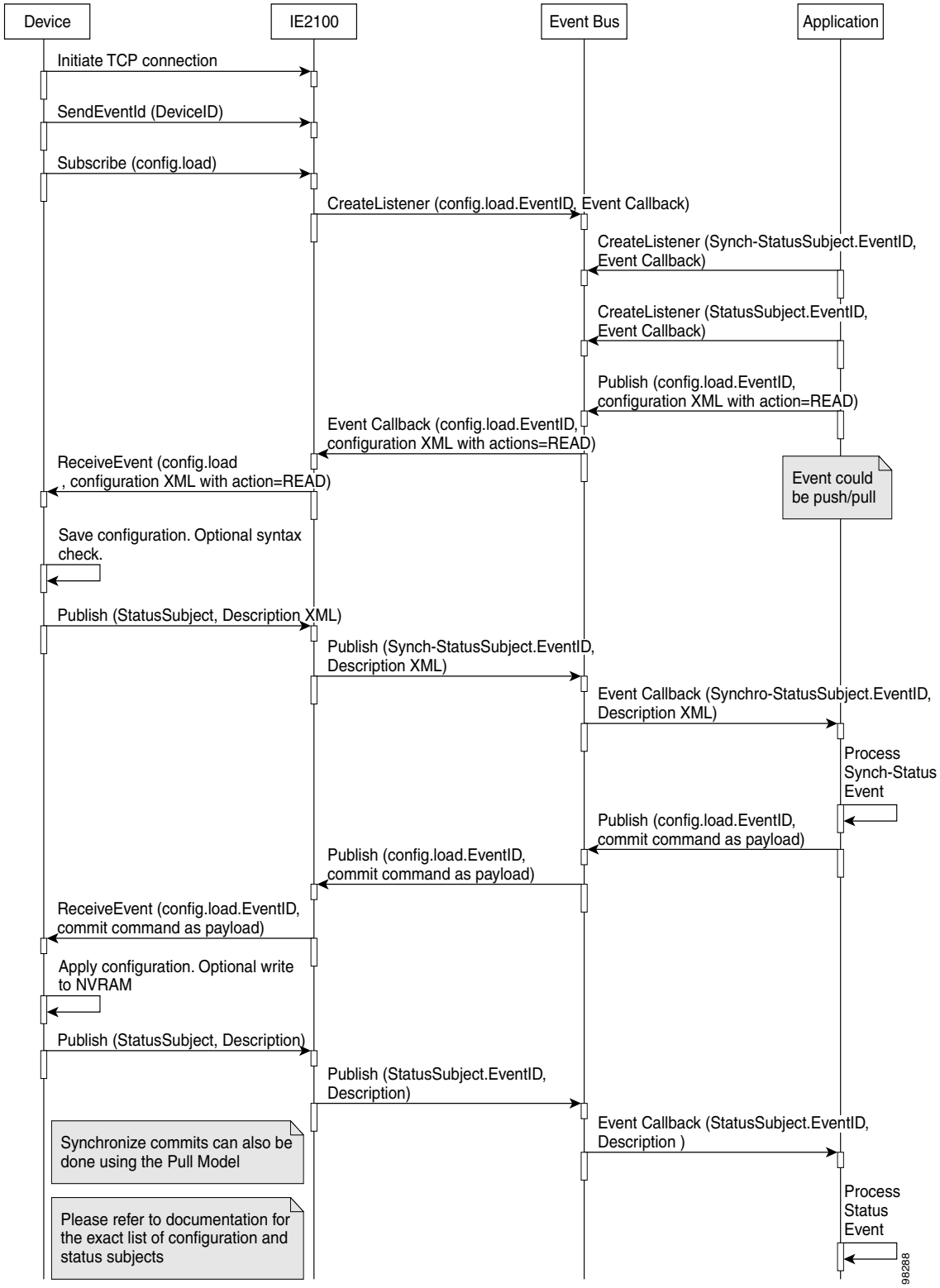
Two stage commit is the process by which an application can send a configuration to the device, but can make it take effect at a later date and time. This is particularly useful for applications that need to synchronize configurations among multiple devices.

Sequence of Operations in Two-stage Commit

- When the event agent begins execution on the device, it initiates a TCP connection to Cisco Configuration Engine, and passes its Event ID, therefore identifying itself by that ID for event-based communication.
- When the configuration agent on the device begins execution, it starts subscribing to a configuration subject (cisco.cns.config.load).
- This subscription request is received by the Event Gateway component of Cisco Configuration Engine, and it creates a listener for the configuration subject, further qualified by the Event ID of that device. It registers an event callback that will be executed when an event is published on the listening subject.
- When an external application has a configuration update to be sent to the device, it starts subscribing to the status subject from the device, and a synchronization status subject, qualified by the Event ID of the device. If this step is not done, the application will lose any status events published from the device.

  The status subject is for the status of the second stage in the two-stage commit, and the synchronization status subject is for the status of the first stage in the process.
- The application then publishes an event on the configuration subject, which can be either a pull or push. But in the payload, the application indicates that the action for this operation is “read”, which means that the device should not apply the configuration right away, but store it for a future commit.
- The Event Callback registered by Cisco Configuration Engine for the configuration subject will be executed.
- The payload of the event will be passed to the device along with the subject.
- The device, upon receiving the event, parses the XML payload, and “reads” in the configuration, without actually applying the configuration commands. It publishes a synchronization status message, which will be available on the Event Bus through Cisco Configuration Engine.
- Since the external application created a listener for the synchronization status subject, the application's event callback will be called, where the status message can be processed.
- If the synchronization status indicated a success, the application may send a commit event to the device, or a cancel if the application wishes to cancel the operation.
- The device will then respond with a status subject, to indicate the success or failure of the synchronization operation.

Figure 8-3 Sequence Diagram for a Two-stage Commit



Creating Application Using SDK for Agent Enabled Devices

To create a provisioning solution using this SDK in a pull operation, you need to perform the following steps:

Step 1 Make sure that you have a Cisco Configuration Engine setup. Refer to the *Cisco Configuration Engine Installation & Configuration Guide, 3.5.3* for further information. If using Internal Directory Mode of the Cisco Configuration Engine, skip to Step 5.

Step 2 If operating in External Directory Mode, install a Directory Server to store device information. Cisco Configuration Engine supports LDAP version 3 and has been tested with NDS, iPlanet, and Active Directory directory services.

Step 3 If using the External Directory Mode, see "Loading the Device Schema in Chapter 1," define the schema for storing the device configuration information.

Step 4 If using the External Directory Mode, apply the device schema to the Directory Server.

Step 5 If using pull mode, create device object in the Cisco Configuration Engine data store.

Step 6 If Namespace Mapper is used, create group, application and event object. Those can be created using the UI.

Step 7 If in External Directory Mode, set up and configure the host system and enter the device and Namespace Mapper schema details. Besides the network configuration, the host also prompts for Directory information. Refer to the *Cisco Configuration Engine Administration Guide* for setup information.

Step 8 Create Device configuration templates. These templates can be stored either on the host system or on an external http server. If you want to store the templates locally on the host system, the template file manager interfaces can be used to import/export/remove the templates. The host system provides an XML interface to manage the templates on the host system.

Once these templates are created, they can be associated to the device by updating the device information through the UI.

Step 9 Each IOS device must be running Cisco IOS image 12.2T or later, and must be set up to talk to the host system. In order for the IOS device to pick the initial configuration from the device, the device must have the configuration agent enabled.

In order to pick up any updates, the IOS device must have the event agent and the partial config agent enabled

Using other options available in the Cisco Configuration Engine commands, user can enable/disable syntax checking, specify a backup event gateway, and specify keepalive time intervals. If the device is not agent enabled, you can skip this step, but instead, you need to set up IMGW either with the IMGW Device Information Database API, or with the Cisco Configuration Engine GUI.

Step 10 Write an application using the Event API to publish and subscribe events on the Event bus. The provisioning application can trigger the device to pick up an updated configuration from the host system. The provisioning application can also push configuration commands to the device directly.

The device publishes all the results on the event bus. The provisioning application should also subscribe to appropriate subject names to pick the results from the device. The message format for all these events is XML and hence there are DTDs corresponding to each event. Refer to Chapter 9, "DTDs for Cisco IOS Devices" for further information.

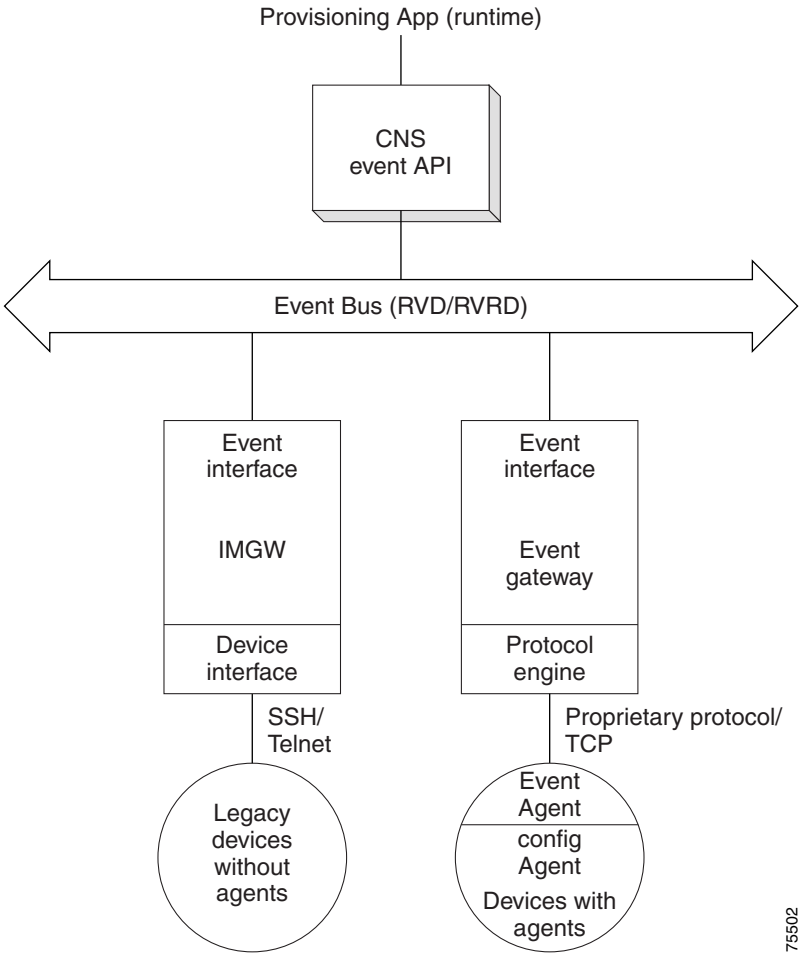
Creating Provisioning Application for Non-Agent Enabled Devices

Using the DID API, create device objects and add hop information for the devices.

If using NSM, create device, group, application and event objects using the UI.

Write an application using the Event API to publish and subscribe events on the Event Bus. The devices can be configured either using pull mode or push mode. The status of the configuration operation will also be published on the event bus and can be processed by the application. The XML DTDs for the configuration and status messages are documented in Chapter 7.

Figure 8-4 IMGW works in concert with the Event Gateway to reach legacy devices without a built-in event agent



The following sections describe the sequence of operations for initial and partial configuration using Cisco Configuration Engine for agent-enabled devices. For the complete list of features offered by Cisco Configuration Engine and their usage, please refer to the official Cisco Configuration Engine documentation.

CHAPTER 9

DTDs for Cisco IOS Devices

Event Gateway Communications

The Event Gateway uses Extensible Markup Language (XML) to communicate with Cisco devices. The Cisco IOS configuration agent uses the common prefix *cisco.mgmt.cns.config* for all subjects that it subscribes for or posts to. The configuration agent subscribes only to cisco.mgmt.cns.config.load. All messages sent TO the agent by an application must be published using this subject name. The other subjects listed below are used for messages sent OUT from the configuration agent.

- *cisco.mgmt.cns.config.load*—to obtain configuration events; the following messages can be sent to the configuration agent:
  - config-server
    config-event
    config-write
- *cisco.mgmt.cns.config.complete*—config-success is published on this subject to post successful completion of the configuration; it is published with the cisco.mgmt.cns,config.sync-status when reporting a failure of a config-write message
- *cisco.mgmt.cns.config.failure*—config-failure is published on this subject to post configuration failure; it is published with the subject cisco.mgmt.cns.config.sync-status when it is reporting a failure of a config-write message.
- *cisco.mgmt.cns.config.warning* — config-warning is published on this subject to post a warning
- *cisco.mgmt.cns.config.sync-status*—used in conjunction with synchronized partial configurations to post success or failure of read and syntax-check operations, so that the application can complete the configuration load

config_id

The *config_id* is a text string passed by the device to the Configuration Engine SDK, where it is used as a directory-lookup key. It is the responsibility of the network administrator to ensure the correlation between *config_id* as it is defined on the device and as it is defined in the directory attribute container corresponding to device's intended configuration.

In the IAD family of Cisco platforms, the *config_id* default value (hostname) is overridden with a string value resolved by the IOS IAD platform subsystem. This IAD subsystem uses a dynamic system whereby each IAD device learns its *config_id* at run-time.

Event DTDs and Sample XML

The following DTDs define the XML used for communications between devices and the Event Gateway.

cisco.mgmt.cns.config.load

The config.load message can take three different forms:

- Push Message carrying CLI in the payload
- Pull Message that causes the router to access a URL to retrieve its config
- Write Message that causes a previously sent config to be applied.

These messages are generated by a user application that wishes a router to modify its config. In response to these messages, the network element replies with one of the Completion Messages. For a 2-phase commit, the exchange is illustrated below.

Push Message

A Push Message is an Bus Message that contains a CLI payload that is pushed onto the Network Element. The Bus subject used is cisco.mgmt.cns.config.load.{device-id} All associated information about this action is contained within the same message. The pushed configuration can be applied in two-phase commit like manner to either the running or to both the running and the startup config.

For a 2-phase commit type of exchange, this message/event needs to be followed by a Write Message with a matching config-id. 2-phase commit is indicated by the attribute config-action having the value read.

DTD for a Push Message

```
<!ELEMENT config-event (identifier, config-data)>
<!ATTLIST config-event config-action (read | write | persist) #REQUIRED no-syntax-check
(TRUE | FALSE) #REQUIRED>
<!ELEMENT config-data (config-id, cli*)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT cli (#PCDATA)>
```

Table 9-1 Elements of a Push Message

XML Element	Description
config-event	This is the root tag for the push event payload. It takes two attributes config-action and no-syntax-check. The config-action defines what the router does when it receives the config. There are three possibilities: write - update only the running config persist - update both the running config and the startup config in NVRAM read - store the new config in a buffer on the router, but do not apply until a second event is sent requesting that the config be applied The no-syntax-check defines whether the config sent to the router is passed through a syntax checker before being applied. There are two options: TRUE - do not syntax check the config FALSE - syntax check the config before it is applied
identifier	This string uniquely identifies this message within the whole NMS/OSS system. Any messages sent in response to this message will have the same identifier set in their payload. A subsequent Write Message needs to mention this string to confirm the commit of the CLI to the Network Element.
config-data	his tag is a container. It includes the identifier of the device to be configured and the CLI that is to be applied to the device.
config-id	This tag identifies the config ID of the device that the CLI is to be applied to. The value must match the ID that has been configured on the device using the cns id command.
CLI	This tag contains an individual line of IOS cli that is to be applied to the device. Multiple instances of this tag can be included in the message to create complex config to be applied.

Sample XML for a Push Message

```
<?xml version "1.0" encoding="UTF-8"?>
<config-event config-action="write" no-syntax-check="TRUE">
    <identifier>sdoeventtest</identifier>
    <config-data>
        <config-id>sdo-3660-1</config-id>
        <cli>interface Loopback123</cli>
        <cli>description loopbacktest</cli>
        <cli>ip address 10.10.10.7 255.255.255.255</cli>
    </config-data>
</config-event>
```

Completion Messages for a Push Message

Any one of the following messages, containing the config-id specified in the Push Message, are possibly returned by the Network Element.

- A Complete Message, indicating successful execution of the config change. When attribute config-action has the value write the config change was applied to the device, when it has the value read, only the syntax-check succeeded.
- A Failure Message, indicating unsuccessful execution of the config change. When attribute config-action has the value write. When attribute config-action has the value read, this indicates that the syntax check failed.

- A Warning Message, indicating partial execution of the config change, returned only when attribute config-action has the value write.

Pull Message

A Pull Message is an Bus Message that contains a URL that will be pulled by the Network Element. No CLI is present for this message, only associated information about this action is contained within the same message. The pulled configuration can be applied in two-phase commit like manner to either the running or to both the running and the startup config.

For a 2-phase commit type of exchange, this message/message needs to be followed by a Write Message with a matching config-id. 2-phase commit is indicated by the attribute config-action having the value read.

DTD for a Pull Message

```
<!ELEMENT config-server (identifier, server-info)>
<!ATTLIST config-server config-action (read | write | persist) #REQUIRED no-syntax-check
(TRUE | FALSE ) #REQUIRED>
<!ELEMENT server-info (ip-address, web-page)>
<!ELEMENT ip-address (#PCDATA)>
<!ELEMENT web-page (#PCDATA)>
<!ELEMENT identifier (#PCDATA)>
```

Table 9-2 XML Elements of Pull Messages

XML Elements	Description
config-server	This is the root tag for the pull message payload. It takes two attributes config-action and no-syntax-check. The config-action defines what the router does when it receives the config. There are three possibilities: write - update only the running config persist - update both the running config and the startup config in NVRAM read - store the new config in a buffer on the router, but do not apply until a second message is sent requesting that the config be applied. The no-syntax-check defines whether the config sent to the router is passed through a syntax checker before being applied. There are two options: TRUE - do not syntax check the config FALSE - syntax check the config before it is applied
identifier	This string uniquely identifies this message within the whole NMS/OSS system. Any messages sent in response to this message will have the same identifier set in their payload. A subsequent Write Message needs to mention this string to confirm the commit of the CLI to the Network Element.
server-info	This tag is a container that includes the IP address of the server and the path on that server to get the config from. Note that this tag and its children are required in the message but they are partially ignored by the device due to security concerns. This functionality may be reactivated in future IOS releases.
ip-address	The IP address of the server from which to download the config. Note that this tag is ignored by the device due to security concerns. This functionality may be reactivated in future IOS releases.
web-page	The path on the server from which to download the config. Note that this tag executed by the device, which might lead to security concerns. This functionality may be changed in future IOS releases.

Sample XML for a Pull Message

```
<?xml version="1.0" encoding="UTF-8"?>
<config-server config-action="write" no-syntax-check="TRUE">
    <identifier>provisionningID-231047</identifier>
    <server-info>
        <ip-address>sdo-ie2100-1.cisco.com</ip-address>
        <web-page>/TelecomSirius/North/Edge-Dev-153/Fa2/1</web-page>
    </server-info>
</config-server>
```

Completion Messages for a Pull Message

Any one of the following messages, containing the config-id specified in the Pull Message, are possibly returned by the Network Element.

- A Complete Message, indicating successful execution of the config change. When attribute config-action has the value write the config change was applied to the device, when it has the value read, only the syntax-check succeeded.
- A Failure Message, indicating unsuccessful execution of the config change. When attribute config-action has the value write. When attribute config-action has the value read, this indicates that the syntax check failed.
- A Warning Message, indicating partial execution of the config change, returned only when attribute config-action has the value write.

Write Message

For a 2-phase commit type of exchange, a Write Message containing a config-id that matches a previously sent Push or Pull Message is used.

DTD for a Write Message

```
<!ELEMENT config-write (identifier)>
<!ATTLIST config-write write-action (write | cancel | persist) #REQUIRED>
<!ELEMENT identifier (#PCDATA)>
```

Table 9-3 XML Elements for Write Messages

XML Elements	Description
config-write	This is the root tag for the write message payload. It takes one attribute write-action. The write-action defines what the router does when it applies the config. There are three possibilities: write - update only the running config persist - update both the running config and the startup config in NVRAM cancel - remove the config from the buffer, cancelling the task. No CLI will be applied.
identifier	This string uniquely identifies this message within the whole NMS/OSS system. Any messages sent in response to this message will have the same identifier set in their payload. The value must match that of a previously sent Push or Pull Message that has it's config queued in the buffer to be applied. On a single network element, there can only be one outstanding configuration change in the buffer.

Sample XML for a Write Message

```
<?xml version="1.0" encoding="UTF-8"?>
<config-write write-action="write">
    <identifier>sdoeventtest</identifier>
</config-write>
```

Completion Messages for a Write Message

Any one of the following messages, containing the config-id specified in a Write Message, are possibly returned by the Network Element. The subject is cisco.mgmt.cns.config.sync-status.

- A Sync-Complete Message, indicating successful execution of the config change. This message indicates successful configuration of the network element.
- A Sync-Failure Message, indicating unsuccessful execution of the config change. The syntax check already took place during the execution of a Push or Pull Message, hence this indicates some other error condition.
- A Warning Message, indicating partial execution of the config change.

cisco.mgmt.cns.config.complete

The cisco.mgmt.cns.config.complete subject is used in a message sent in response to a message with cisco.mgmt.cns.config.load subject when the config has been successfully applied. This modification applies to either the running-config or both the running-config and the startup-config dependent on the attribute config-action. If the syntax check option had been enabled in the cisco.mgmt.cns.config.load message then the config has passed the syntax check and has been successfully applied.

DTD for a Complete Message

```
<!ELEMENT config-success (identifier, config-id)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
```

Table 9-4 XML Elements for a 'complete' Message

XML Elements	Description
config-success	The root tag for the complete message.
identifier	This is a string that uniquely identifies this message within the NMS/OSS system. It will be set to match the identifier set in the load message.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.

Sample XML for a Complete Event

```
<config-success>
    <identifier>sdoeventtest</identifier>
    <config-id>sdo-3660-1</config-id>
</config-success>
```

cisco.mgmt.cns.config.failure

The failure event is sent when either the config fails to pass the syntax check, if this option has been enabled in the cisco.mgmt.cns.config.load message, or the config has failed to be applied. If the failure is during syntax checking then no changes are made to the running-config. If the failure is during the application of the config then some config may have been applied.

DTD for a Failure Event

```
<!ELEMENT config-failure (identifier, config-id, error-info)>
<!ELEMENT error-info (line-number?, error-message)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT line-number (#PCDATA)>
<!ELEMENT error-message (#PCDATA)>
```

Table 9-5 XML Elements for a 'failure' Event

XML Elements	Description
config-failure	The root tag for the failure event
error-info	A container tag that holds the line number tag and error message tag.
identifier	This is a string that uniquely identifies this message within the system. It will be set to match the identifier set in the load event.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.
line-number	The number of the line in the config sent to the device that has failed.
error-message	The error message generated when applying the config failed.

Sample XML for a Failure Event

```
<?xml version="1.0" encoding="UTF-8" ?>
<exec-cmd-event>
    <identifier-exec>show123456</identifier>
    <event-response>cisco.mgmt.cns.exec.rsp.application-dirk1</event-response>
    <cli-exec>show running-config</cli-exec>
</exec-cmd-event>
```

cisco.mgmt.cns.config.warning

Warnings messages are sent when the config is syntactically correct and has been applied, but there is a problem with the semantics of the config. A typical example would be: "IP address already in use on another interface".

DTD for a Warning Message

```
<!ELEMENT config-warning (identifier, config-id, warning-message)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT warning-message (#PCDATA)>
```

Table 9-6 Warning Message argument descriptions

XML Elements	Description
config-warning	The root tag for the warning event
identifier	This is a string that uniquely identifies this message within the system. It will be set to match the identifier set in the load event.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.
warning-message	The warning message generated when applying the config.

Sample XML for a Warning Message

```
<config-warning>
    <identifier>identifierWRITE</identifier>
    <config-id>config_idWRITE</config-id>
    <warning-message> **CLI Line # 5: 10.1.7.0 overlaps with
    Ethernet1/0 **CLI Line # 5: Ethernet1/1: incorrect IP address
    assignment</warning-message>
</config-warning>
```

cisco.mgmt.cns.config.sync-status

There are three different messages that can be sent under the subject cisco.mgmt.cns.config.sync-status, these are config-success, config-failure, and config-warning. These messages are sent in response to two phase commit load messages instead of using the subjects cisco.mgmt.cns.config.<complete|failure|warning>. Two phase commit messages use the subject cisco.mgmt.cns.config.load and carry an attribute config-action with as value read.

DTD for a Sync-complete Message

```
<!ELEMENT config-success (identifier, config-id)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
```

Table 9-7 XML Elements for a 'sync-complete' Message

XML Elements	Description
config-success	The root tag for the complete event.
identifier	This is a string that uniquely identifies this message within the system. It will be set to match the identifier set in the load event.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.

Sample XML for a Sync-complete Message

```
<config-success>
    <identifier>sdoeventtest</identifier>
    <config-id>sdo-3660-1</config-id>
</config-success>
```

DTD for a Sync-failure Message

```
<!ELEMENT config-failure (identifier, config-id, error-info)>
<!ELEMENT error-info (line-number?, error-message)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT line-number (#PCDATA)>
<!ELEMENT error-message (#PCDATA)>
```

Table 9-8 XML Elements for 'sync-complete' Message

XML Elements	Description
config-failure	The root tag for the failure event
error-info	A container tag that holds the line number tag and error message tag.
identifier	This is a string that uniquely identifies this message within the system. It will be set to match the identifier set in the load event.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.
Line-number	The number of the line in the config sent to the device that has failed.
error-message	The error message generated when applying the config failed.

Sample XML for a Sync-failure Message

```
<config-failure>
    <identifier>sdoeventtes</identifier>
    <config-id>sdo-3660-1</config-id>
    <error-info>
        <line-number>2</line-number>
        <error-message>Line number = 2, Error = 2</error-message>
    </error-info>
</config-failure>
```

DTD for a Sync-warning Message

```
<!ELEMENT config-warning (identifier, config-id, warning-message)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT warning-message (#PCDATA)>
```

Table 9-9 XML Elements for 'sync-w2arning' Message

XML Elements	Description
config-warning	The root tag for the warning event
identifier	This is a string that uniquely identifies this message within the system. It will be set to match the identifier set in the load event.

Table 9-9 XML Elements for 'sync-w2arning' Message

XML Elements	Description
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value will match the ID that has been configured on the device using the cns id command.
warning-message	The warning message generated when applying the config.

Sample XML for a Sync-warning Message

```
<config-warning>
    <identifier>identifierWRITE</identifier>
    <config-id>config_idWRITE</config-id>
    <warning-message> **CLI Line # 5: 10.1.7.0 overlaps with
    Ethernet1/0 **CLI Line # 5: Ethernet1/1: incorrect IP address
    assignment</warning-message>
</config-warning>
```

cisco.mgmt.cns.event.boot

The boot event is sent when a device restarts and connects to the Event Gateway. This event has no payload. It is the result of the execution of the a cns config initial CLI command.

cisco.mgmt.cns.device.connect

A connect event is sent whenever a device connects to the Event Gateway. This is the result of a cns event CLI command.

DTD for a Connect Message

```
<!ELEMENT device-connect (Device-id, gateway-ip, gateway-hostname, gateway-port)>
<!ELEMENT device-id (#PCDATA)>
<!ELEMENT gateway-hostname (#PCDATA)>
<!ELEMENT gateway-ip (#PCDATA)>
<!ELEMENT gateway-port (#PCDATA)>
```

Table 9-10 XNL Elements for a 'connect' Message

XML Elements	Description
device-connect	The root tag for the connect event.
device-id	This tag identifies the event ID of the device that has connected. The value will match the ID that has been configured on the device using the cns id CLI command
device-ip	The IP address of the device that connected to the gateway. [[added by Dirk]]
device-port	The port number on the device of the TCP session that makes up the connection.
gateway-hostname	The fully qualified domain name of the host to which the device has connected.

Table 9-10 XNL Elements for a 'connect' Message

XML Elements	Description
gateway-ip	The IP address of the host to which the device has connected.
gateway-port	The port number on the host to which the device has connected.

Sample XML for a Device Connect Message

```
<?xml version="1.0" encoding="UTF-8"?>
<device-connect>
    <device-id>sdo-3660-1</device-id>
    <gateway-hostname>sdo-ie2100-3.cisco.com</gateway-hostname>
    <gateway-ip>10.10.1.1</gateway-ip>
    <gateway-port>11011</gateway-port>
</device-connect>
```

cisco.mgmt.cns.device.disconnect

This event is sent whenever a device disconnects from the Event Gateway.

DTD for a Disconnect Message

```
<!ELEMENT device-disconnect (Device-id, gateway-ip, gateway-hostname, gateway-port)>
<!ELEMENT device-id (#PCDATA)>
<!ELEMENT gateway-hostname (#PCDATA)>
<!ELEMENT gateway-ip (#PCDATA)>
<!ELEMENT gateway-port (#PCDATA)>
```

Table 9-11 XML Elements for a 'disconnect' Message

XML Elements	Description
device-disconnect	The root tag for the disconnect event.
device-id	This tag identifies the event ID of the device that has disconnected. The value will match the ID that has been configured on the device using the cns id command
device-ip	The IP address of the device that connected to the gateway. [[added by Dirk]]
device-port	The port number on the device of the TCP session that makes up the connection.
gateway-hostname	The fully qualified domain name of the host from which the device has disconnected.
gateway-ip	The IP address of the host from which the device has disconnected.
gateway-port	The port number from which the device has disconnected on the host.

Sample XML for a Device Disconnect Event

```
<?xml version="1.0" encoding="UTF-8"?>
<device-disconnect>
    <device-id>sdo-3660-1</device-id>
    <gateway-ip>0.0.0.0</gateway-ip>
```

```
    <gateway-hostname>sdo-ie2100-3.cisco.com</gateway-hostname>
    <gateway-port>11011</gateway-port>
</device-disconnect>
```

cisco.mgmt.cns.exec.cmd

These messages contain CLI that is to executed at the enable prompt and do not modify the config. It should be noted that commands that require user interaction are not permitted. The obvious example is the show command.

DTD for an Exec Event

```
<!ELEMENT exec-cmd-event (identifier-exec,(event-response|server-response)?, cli-exec)>
<!ELEMENT identifier-exec (#PCDATA)>
<!ELEMENT event-response (#PCDATA)>
<!ELEMENT server-response (ip-address-exec,port-number?,url)>
<!ELEMENT ip-address-exec (#PCDATA)>
<!ELEMENT port-number (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT cli-exec (#PCDATA)>
```

Table 9-12 XML Elements for an 'exec' Message

XML Elements	Description
exec-cmd-event	The root tag for the CLI exec event.
identifier-exec	This is a string that uniquely identifies this message within the system. Any events sent in response to this event will have the same identifier set in their payload.
device-id	This tag identifies the event ID of the device that has connected. The value will match the ID that has been configured on the device using the cns id CLI command
event-response	The subject on which the response should be sent. The event ID of the device will be appended.
server-response	A container tag to hold information regarding an HTTP post based response to the exec event.
ip-address-exec	The IP address of the server to post the response to.
port-number	The port number on the server to post the response to.
url	The URL on the server to post the response to.
cli-exec	The CLI command to exec on the device.

Sample XML for an Exec Message

```
<?xml version="1.0" encoding="UTF-8" ?>
<exec-cmd-event>
    <identifier-exec>show123456</identifier>
    <event-response>
        <reply-subject>cisco.mgmt.cns.exec.rsp.application-dirk1</reply-subject>
    </event-response>
    <cli-exec>show running-config</cli-exec>
```

cisco.mgmt.cns.exec.rsp

These events contain the output generated by previous exec events.

DTD for an Exec Response Message

```
<!ELEMENT exec-cmd-response (identifier-exec, status, (error-info|response))>
<!ELEMENT identifier-exec (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT response (#PCDATA)>
<!ELEMENT error-info (line-number?, error-message)>
<!ELEMENT error-message (#PCDATA)>
```

Table 9-13 XML Elements for an 'exec response' Message

XML Elements	Description
exec-cmd-response	The root tag for the CLI exec response event.
identifier-exec	This is a string that uniquely identifies this message within the system. It is set to match the identifier set in the original exec event.
status	Whether the CLI has been successfully executed.
response	The output of the CLI command executed.
error-info	A container tag to hold information regarding to errors generated while executing the CLI.
line-number	The line number of the CLI command that failed.
error-message	The error message output when the command failed.

Sample XML for an Successful Exec Response Event

```
<exec-cmd-response>
    <identifier-exec>sdo-3660-1</identifier-exec>
    <status>success</status>
    <response>*00:09:05.183 UTC Mon Mar 1 1993</response>
</exec-cmd-response>
```

Sample XML for an Failure Exec Response Event

```
<exec-cmd-response>
    <identifier-exec>sdo-3660-1</identifier-exec>
    <status>fail</status>
    <error-info>
        <error-message>UNSUPPORTED_CLI_EXEC_CMD</error-message>
```

```
    </error-info>
</exec-cmd-response>
```

cisco.mgmt.cns.inventory.get

This event is used to generate a dump of the physical inventory of the device. This event has no payload.

cisco.mgmt.cns.inventory.device-details

This event contains all of the details of the hardware that makes up a device. It contains information about both the chassis and the line cards.

DTD for an Inventory Response Message

```
<!ELEMENT device-details (config-id?, connect-interface?, chassis-desc?,
chassis-serialnumber?, chassis-midplane-version?, card-info*>
<!ELEMENT config-id (#PCDATA)>
<!ELEMENT connect-interface (#PCDATA)>
<!ELEMENT chassis-desc (#PCDATA)>
<!ELEMENT chassis-serialnumber (#PCDATA)>
<!ELEMENT chassis-midplane-version (#PCDATA)>
<!ELEMENT card-info
(card-type,card-desc?,slot?,daughter?,serial-number?,part-number?,hw-version?,board-revisi
on?,
ports?,controller?,rma-number?,test-history?,eeprom-size?,eeprom-data?,interface*,controll
er?,voice-port?, card-info*)>
<!ELEMENT card-type (#PCDATA)>
<!ELEMENT card-desc (#PCDATA)>
<!ELEMENT slot (#PCDATA)>
<!ELEMENT daughter (#PCDATA)>
<!ELEMENT serial-number (#PCDATA)>
<!ELEMENT part-number (#PCDATA)>
<!ELEMENT hw-version (#PCDATA)>
<!ELEMENT board-revision (#PCDATA)>
<!ELEMENT ports (#PCDATA)>
<!ELEMENT controller (#PCDATA)>
<!ELEMENT rma-number (#PCDATA)>
<!ELEMENT test-history (#PCDATA)>
<!ELEMENT eeprom-size (#PCDATA)>
<!ELEMENT eeprom-data (#PCDATA)>
<!ELEMENT interface (#PCDATA)>
<!ELEMENT controller (#PCDATA)>
<!ELEMENT voice-port (#PCDATA)>
```

Table 9-14 XML Elements for an 'inventory response' Message

XML Elements	Description
device-details	The root tag for the inventory response event.
config-id	This tag identifies the config ID of the device that the cli is to be applied to. The value must match the ID that has been configured on the device using the cns id command.
connect-interface	The interface on the device through which it connected to the host.
chassis-desc	The name of the chassis type.

Table 9-14 XML Elements for an 'inventory response' Message

XML Elements	Description
chassis-serialnumber	The serial number of the chassis.
chassis-midplane-version	If the chassis has a midplane, the version number of that midplane.
card-info	A container tag to hold information about a particular card in the chassis.
card-type	A numeric identifier for the card type.
card-desc	A description of the card type.
slot	The slot in the chassis that the card is occupying.
ports	The number of ports on the card.
daughter	If a WIC the daughter slot the card is present in.
hw-version	The major and minor engineering version numbers.
board-revision	The revision number of the card, used to track assembly versions.
serial-number	The serial number of the card.
part-number	The 800 part number of the card.
card-connector	If it is a daughter card then describe the type of connector, PCI or WAN Module.
test-history	Used to capture the field failure history.
rma-number	Used to tracking field returns.
eeprom-size	The size of the EEPROM data.
eeprom-data	A hex dump of the EEPROM data.
interface	The name of the interface on the card.
controller	The name of the controller on the card.
voice-port	The name of the voice port on the card.

Sample XML for an Inventory Response Message

```
<device-details>
    <chassis-desc>3660</chassis-desc>
    <chassis-serialnumber>JAB0450C03B</chassis-serialnumber>
    <card-info>
        <card-type>179</card-type>
        <card-desc>C3600 Mother board 2FE(TX)</card-desc>
        <slot>0</slot>
        <ports>2</ports>
        <board-revision>A0</board-revision>
        <serial-number>JAB045001AA</serial-number>
        <part-number>800-04737-04</part-number>
        <rma-number>255-255-255-255</rma-number>
        <eeprom-size>128</eeprom-size>
        <eeprom-data>04 FF C1 8B 4A 41 42 30 34 35 30 30 31 41 41 09
        34 40 00 B3 C0 46 03 20 00 12 81 04 42 41 30 85 1C 0C A2 02
        80 FF FF FF FF C4 08 FF FF FF FF FF FF FF FF 81 FF FF FF FF
        03 FF 04 FF C5 08 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF 00 </eeprom-data>
```

```
    <interface>FastEthernet0/0</interface>
    <interface>FastEthernet0/1</interface>
</card-info>
<card-info>
    <card-type>84</card-type>
    <card-desc>Mueslix-4T</card-desc>
    <slot>1</slot>
    <ports>4</ports>
    <hw-version>1.1</hw-version>
    <board-revision>H0</board-revision>
    <serial-number>22737613 </serial-number>
    <part-number>800-02314-02</part-number>
    <rma-number>00-00-00</rma-number>
    <eeprom-size>128</eeprom-size>
    <eeprom-data>01 54 01 01 01 5A F2 CD 50 09 0A 02 00 00 00 00
    88 00 00 00 00 12 01 00 00 05 FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF </eeprom-data>
    <interface>Serial1/0</interface>
    <interface>Serial1/1</interface>
    <interface>Serial1/2</interface>
    <interface>Serial1/3</interface>
</card-info>
<card-info>
    <card-type>67</card-type>
    <card-desc>Ethernet</card-desc>
    <slot>2</slot>
    <ports>1</ports>
    <hw-version>1.1</hw-version>
    <board-revision>E0</board-revision>
    <serial-number>19662010 </serial-number>
    <part-number>800-02026-03</part-number>
    <rma-number>00-00-00</rma-number>
    <eeprom-size>128</eeprom-size>
    <eeprom-data>01 43 01 01 01 2C 04 BA 50 07 EA 03 00 00 00 00
    70 00 00 00 00 05 10 00 FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF </eeprom-data>
    <interface>Ethernet2/0</interface>
</card-info>
<card-info>
    <card-type>67</card-type>
    <card-desc>Ethernet</card-desc>
    <slot>3</slot>
    <ports>1</ports>
    <hw-version>1.1</hw-version>
    <board-revision>E0</board-revision>
    <serial-number>19661982 </serial-number>
    <part-number>800-02026-03</part-number>
    <rma-number>00-00-00</rma-number>
    <eeprom-size>128</eeprom-size>
    <eeprom-data>01 43 01 01 01 2C 04 9E 50 07 EA 03 00 00 00 00
    70 00 00 00 00 05 10 00 FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF FF FF FF FF FF FF FF FF FF </eeprom-data>
```

```
        <interface>Ethernet3/0</interface>
    </card-info>
    <card-info>
        <card-type>67</card-type>
        <card-desc>Ethernet</card-desc>
        <slot>4</slot>
        <ports>1</ports>
        <hw-version>1.1</hw-version>
        <board-revision>E0</board-revision>
        <serial-number>19661861 </serial-number>
        <part-number>800-02026-03</part-number>
        <rma-number>00-00-00</rma-number>
        <eeprom-size>128</eeprom-size>
        <eeprom-data>01 43 01 01 01 2C 04 25 50 07 EA 03 00 00 00 00
        70 00 00 00 00 05 10 00 FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
        FF FF FF FF FF FF FF FF FF FF FF FF </eeprom-data>
        <interface>Ethernet4/0</interface>
    </card-info>
    </device-details>
```

cisco.mgmt.cns.event.id-changed

This event is sent whenever the device ID of a device that is connected to the Event Gateway is changed. This is the result of a cns id CLI command.

DTD for an Event ID Changed Message

```
<!ELEMENT event-id-changed (time, new_data, old_data)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT new_data (#PCDATA)>
<!ELEMENT old_data (#PCDATA)>
```

Table 9-15 XML Elements for an 'ID changed' Message

XML Elements	Description
event-id-changed	The root tag for the event ID changed event.
Time	The time that the event ID was changed.
new-data	The new event ID.
old-data	The old event ID.

Sample XML for an Event ID Changed Event

```
<event-id-changed>
    <time>03:37:40 BST Mon Jul 29 2002</time>
    <new_data>sdo-3660-1</new_data>
    <old_data>test-3660-1</old_data>
</event-id-changed>
```

cisco.mgmt.cns.config.id-changed

This event is sent when ever the config ID of a device that is connected to the Event Gateway is changed.

DTD for an Config ID Changed Event

```
<!ELEMENT config-id-changed (time, new_data, old_data)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT new_data (#PCDATA)>
<!ELEMENT old_data (#PCDATA)>
```

Table 9-16 XML Elements for a 'config-changed' Message

XML Elements	Description
config-id-changed	The root tag for the config ID changed event.
time	The time that the config ID was changed.
new-data	The new config ID.
old-data	The old config ID.

Sample XML for an Config ID Changed Event

```
<config-id-changed>
    <time>03:37:40 BST Mon Jul 29 2002</time>
    <new_data>sdo-3660-1</new_data>
    <old_data>test-3660-1</old_data>
</event-id-changed>
```

cisco.mgmt.cns.config-changed

This event is sent when the running config on the network element is changed. It is equivalent to the syslog message "config changed by console on console".

DTD for a Config-changed Event

```
<!ELEMENT config-changed (time, new_data, old_data)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT new_data (#PCDATA)>
<!ELEMENT old_data (#PCDATA)>
```

Table 9-17 XML Elements for a 'config-changed' Message

XML Elements	Description
config-changed	The root tag for the config ID changed event.
time	The time that the config ID was changed.
new-data	The new config or NULL if no new config.
old-data	The old config or NULL if no old config.

Sample XML for a config-changed event

Sample DTD for Config-changed Event

```
<!ELEMENT cli (#PCDATA)>
<!ELEMENT cnsMessage (senderCredentials, messageId, notify)>
<!ATTLIST cnsMessage
        version CDATA #REQUIRED
>
<!ELEMENT senderCredentials (userName, (passWord?))>
<!ELEMENT messageId (#PCDATA)>
<!ELEMENT notify (configChanged)>
<!ELEMENT userName (#PCDATA)>
<!ELEMENT passWord (#PCDATA)>
<!ELEMENT configChanged (sequence, (lostData | (changeInfo, changeData)))>
<!ATTLIST configChanged
        version CDATA #IMPLIED
>
<!ELEMENT sequence (#PCDATA)>
<!ATTLIST sequence
        lastReset CDATA #REQUIRED
>
<!ELEMENT lostData ANY>
<!ELEMENT changeInfo (user?, (async | telnet | cns), when)>
<!ELEMENT changeData (changeItem)+>
<!ELEMENT user (#PCDATA)>
<!ELEMENT async (port)>
<!ELEMENT telnet (srcIP, vtyName)>
<!ELEMENT cns (identifier)>
<!ELEMENT when (absoluteTime | sysUptime)>
<!ELEMENT changeItem (context, enteredCommand, oldConfigState,
newConfigState)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT srcIP (#PCDATA)>
<!ELEMENT vtyName (#PCDATA)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT absoluteTime (#PCDATA)>
<!ELEMENT sysUptime (#PCDATA)>
<!ELEMENT context (cli)*>
<!ELEMENT enteredCommand (cli)+>
<!ELEMENT oldConfigState (cli)*>
<!ELEMENT newConfigState (cli)*>
```

Sample XML

```
<?xml version="1.0" encoding="UTF-8"?>
<cnsMessage version="1.0">
        <senderCredentials>
                <userName>dvlpr-7200-2</userName>
                <passWord>2299865183008923422</passWord>
        </senderCredentials>
        <messageId>dvlpr-7200-2_4</messageId>
        <notify>
                <configChanged version="1.0">
                        <sequence
lastReset="2003-04-23T20:27:03.071Z">2</sequence>
                        <changeInfo>
                                <async>
                                        <port>con_0</port>
                                </async>
                                <when>
```

```
<absoluteTime>2003-04-23T20:27:19.847Z</absoluteTime>
                                </when>
                        </changeInfo>
                        <changeData>
                                <changeItem>
                                        <context></context>
                                        <enteredCommand>
                                                <cli>cns exec 10.1.3.3</cli>
                                        </enteredCommand>
                                        <oldConfigState></oldConfigState>
                                        <newConfigState>
                                                <cli>cns exec  10.1.3.3
80</cli>
                                        </newConfigState>
                                </changeItem>
                                <changeItem>
                                        <context></context>
                                        <enteredCommand>
                                                <cli>cns exec 10.1.3.3</cli>
                                        </enteredCommand>
                                        <oldConfigState></oldConfigState>
                                        <newConfigState>
                                                <cli>cns exec  10.1.3.3
80</cli>
                                        </newConfigState>
                                </changeItem>
                        </changeData>
                </configChanged>
        </notify>
</cnsMessage>
```

Sample XML for Lost-changes Event

```
<?xml version="1.0" encoding="UTF-8"?>
<cnsMessage version="1.0">
        <senderCredentials>
                <userName>dvlpr-7200-2</userName>
                <passWord>2299865183008923422</passWord>
        </senderCredentials>
        <messageId>dvlpr-7200-2_24</messageId>
        <notify>
                <configChanged version="1.0">
                        <sequence
lastReset="2003-04-25T00:49:01.769Z">11</sequence>
                        <lostData></lostData>
                </configChanged>
        </notify>
</cnsMessage>
```

cisco.mgmt.cns.snmp.rqst

These messages contain the base64 encoded value of a non-granular snmp request.

DTD for a non-granular snmp request message

```
<!ELEMENT SnmpMessage (#PCDATA)>
```

Table 9-18

Element	Description
SnmpMessage	The root tag of the message and contains the base64 encoded value of the snmp request pdu

example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SnmpMessage>MCsCAQAEBnB1YmxpY6AeAgRTeSCiAgEAAgEAMBAwDgYKKwYBAgECAgEHAQUA</SnmpMessage>
```

cisco.mgmt.cns.snmp.resp

These messages contain the base64 encoded value of a non-granular snmp response.

DTD for a non-granular snmp response message

```
<!ELEMENT SnmpMessage (#PCDATA)>
```

Table 9-19

Element	Descriptions
SnmpMessage	The root tag of the message and contains the base64 encoded value of the snmp response pdu.

example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SnmpMessage>MCwCAQAEBnB1YmxpY6IfAgRTeSCiAgEAAgEAMBEwDwYKKwYBAgECAgEHAQIBAQ==</SnmpMessage
```

cisco.mgmt.cns.snmp.trap

These messages contain the base64 encoded value of a non-granular snmp trap/notification.

DTD for a non-granular snmp response message

```
<!ELEMENT SnmpMessage (#PCDATA)>
```

Table 9-20 Non-granular SnmpMessage element

Element	Descriptions
SnmpMessage	The root tag of the message and contains the base64 encoded value of the snmp trap/notification pdu.

example:


```
<?xml version="1.0" encoding="UTF-8" ?>
<SnmpMessage>MHoCAQAEBnB1YmxpY6RtBgkrBgEEAQkJKwJAE2Npc2NvLmNucy5
zbm1wLnRyYXACAQYCAQFDBAH0m6owPzATBg4rBgEEAQkJKwEBBgEDCQIBATATBg4rBgEEAQk
JKwEBBgEECQIBAjATBg4rBgEEAQkJKwEBBgEFCQIBAw==
</SnmpMessage>
```


cisco.mgmt.cns.mibaccess.request

These messages contain the granular snmp Get/Set operations.

DTD for a granular mibaccess request message

```
<!ELEMENT Get (Getvarbind)+>
    <!ATTLIST Get
    device-id CDATA #REQUIRED
    request-id CDATA #REQUIRED
    >
    <!ELEMENT Set (Setvarbind)+>
    <!ATTLIST Set
    device-id CDATA #REQUIRED
    request-id CDATA #REQUIRED
    >
    <!ELEMENT Getvarbind EMPTY>
    <!ATTLIST Getvarbind
    oid CDATA #REQUIRED
    >
    <!ELEMENT Setvarbind (#PCDATA)>
    <!ATTLIST setvarbind
        oid CDATA #REQUIRED
        type (int | int32 | uint32 | ip | counter32 | counter64 | gauge32 | timeticks
            | opaque | bits | oid | octetstring) #REQUIRED
    >
```

Table 9-21 Elements for a granular mibaccess request message.

Element	Descriptions
Get	The root tag of a snmp get request. Perform a snmp get on the listed varbinds. *device-id* contains the device id. *request-id* has the request identifier.
Getvarbind	The varbind to retrieve. *oid* contains the object ID of the MIB object.

Table 9-21 Elements for a granular mibaccess request message.

Element	Descriptions
Set	The root tag of a snmp set request. Perform a snmp set on the listed varbinds. *device-id* contains the device id. *request-id* has the request identifier.
Setvarbind	The varbind to set. *oid* contains the object ID of the MIB object. *type* is the type of the object. For types int, int32, uint32, couter32, counter64, gauge32 adn timeticks, value is in string form of decimal representation. For types opaque, bits, ip, octetstrings and any, it is the base64 encoded value of the string of octets. For type oid, it is in the form "a.b.c.d..."

examples:

Example of XML data to get all the ifEntries:

```
<?xml version="1.0" encoding="UTF-8"?>
<Get request-id="100"  device-id="event_3600_1">
  <Getvarbind oid="1.3.6.1.2.1.2.2.1"></Getvarbind>
</Get>
```

Example of XML data to get all the instances of ifDescr Object:

```
<?xml version="1.0" encoding="UTF-8"?>
<Get device-id = “Networkmanager1” request-id = “200”>
  <Getvarbind oid= “1.3.6.1.2.1.2.2.1.2 “/>
</Get>
```

Example of XML data to set instance of ifAdminStatus.2 to ë1

```
<?xml version="1.0" encoding="UTF-8"?>
<Set device-id = “Networkmanager1” request-id = “400”>
  <Setvarbind oid= “1.3.6.1.2.1.2.2.1.7.2.1” > 1</Setvarbind>
</Set>
```

cisco.mgmt.cns.mibaccess.response

These messages contain the granular snmp responses from Get/Set operations.

DTD for a granular mibaccess response message

```
<!ELEMENT Response (Varbind)*>
    <!ATTLIST Response
    request-id CDATA #REQUIRED
    fragment-id CDATA #REQUIRED
        error-status(genError | wrongValue | noCreation | inconsistentValue |
resourceUnavailable
          | notWritable | wrongType | inconsistentName | noSuchObject | noSuchInstance
          | endOfMibView|noError) PCDATA #REQUIRED
    error_index CDATA #REQUIRED>

    <!ELEMENT Varbind (#PCDATA)>
    <!ATTLIST Varbind
```

```
        name PCDATA #REQUIRED
        type (int | int32 | uint32 | ip | counter32 | counter64 | gauge32 | timeticks |
opaque
           | bits | oid | octetstring) #REQUIRED
        error (genError | wrongValue | noCreation | inconsistentValue |
resourceUnavailable
           | notWritable | wrongType
           | inconsistentName | noSuchObject | noSuchInstance | endOfMibView|noError)
#IMPLIED>
```

Table 9-22 Elements for a granular mibaccess response message.

Element	Descriptions
Response	The root tag of the response event. *request-id* is the request identifier. *fragment-id* is the fragment id of the response when the response exceeds some specific size. The first fragment is 1. *error_index* is the first object that cannot be set. *error_status* contains the error that occurred in setting that object referenced by the error_index.
Varbind	The value of the *oid* in this varbind. *name* is the object ID and is formatted in text string. *type* is the type of the object. For types int, int32, uint32, couter32, counter64, gauge32 adn timeticks, value is in string form of decimal representation. For types opaque, bits, octetstrings and any, it is the base64 encoded value of the string of octets. For types ip and oid, it is in the form "a.b.c.d..."

example:

```
<Response device-id = “NetworkManager1” request-id = “300”>
   <varbind name = “1.3.6.1.3.1099.1.4.3.1.3.101.102.103.104.105.106” type = “oid”> 0.0
</varbind>
   <varbind name = “1.3.6.1.3.1099.1.4.3.1.2.3.101.102.103.104.105.106” type =
“octetstring”> b3duZXI=
   </varbind>
</Response>
```

cisco.mgmt.cns.mibaccess.notification

These messages contain the granular snmp trap/notification.

DTD for a granular mibaccess notification message

```
<!ELEMENT Notification (Varbind)*>
    <!ATTLIST Notification
    timestamp CDATA #REQUIRED
    notificationOID CDATA #REQUIRED
    >
    <!ELEMENT Varbind (#PCDATA)>
    <!ATTLIST Varbind
        name PCDATA #REQUIRED
```

```
        type (int | int32 | uint32 | ip | counter32 | counter64 | gauge32 | timeticks |
opaque
           | bits | oid | octetstring) #REQUIRED
        error (genError | wrongValue | noCreation | inconsistentValue |
resourceUnavailable
           | notWritable | wrongType
           | inconsistentName | noSuchObject | noSuchInstance | endOfMibView|noError)
#IMPLIED>
```

Table 9-23 Elements for a granular mibaccess notification message.

Element	Descriptions
Notification	The root tag for the notification *event. timestamp* contains time of the *notification. notificationOID* contains object ID of interest.
Varbind	The value of the object ID in this varbind. *name* is the object ID and is formatted in text string. *type* is the type of the object. For types int, int32, uint32, couter32, counter64, gauge32 adn timeticks, value is in string form of decimal representation. For types opaque, bits, octetstrings and any, it is the base64 encoded value of the string of octets. For types ip and oid, it is in the form "a.b.c.d..."

example:

```
[2003-10-21 14:06:56]:
<?xml version="1.0" encoding="UTF-8"?>
<Notification notificationOID="1.3.6.1.4.1.9.9.43.2.0.1" timestamp="32807850">
   <Varbind name="1.3.6.1.4.1.9.9.43.1.1.6.1.3.9" type="int"
     error="noError">1</Varbind>
   <Varbind name="1.3.6.1.4.1.9.9.43.1.1.6.1.4.9" type="int"
     error="noError">2</Varbind>
   <Varbind name="1.3.6.1.4.1.9.9.43.1.1.6.1.5.9" type="int"
     error="noError">3</Varbind>
</Notification>
```

CHAPTER 10

IMGW API Reference

This chapter describes the information related to IMGW API.

Note The IMGW API provided in *IMGWDeviceClient.jar* has been deprecated. These APIs do not work together with the new Device Admin API, or the Admin Web Service APIs.

IMGWDevice API

package com.cisco.cns.imgw

public class IMGWDevice

A class used to access the IMGW deviceinfo database.

Public Data Structures and Types

static final int IMAGE_AGENT

Usage:

```
static final int IMAGE_AGENT;
```

static final int CONFIG_AGENT

Usage:

```
static final int CONFIG_AGENT;
```

Public Methods

setDebug()

A public method to set the m_debug_enable member data.

Usage:

```
void setDebug(boolean myDebug);
```

Parameters

myDebug—boolean variable

IMGWDevice()

A public constructor of IMGWDevice.

Usage:

```
IMGWDevice(String url) throws NetworkException;
```

Parameters

url—the URL where IMGWServlet runs.

Exceptions

NetworkException—if the specified URL is malformed

addErrorPattern()

The method to add the pattern to be recognized as error for the particular device type.

Usage:

```
void addErrorPattern(String devType,
     String pattern) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type.

pattern—pattern string being added.

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

deleteErrorPattern()

The method to delete the pattern from the error list for the particular device type.

Usage:

```
void deleteErrorPattern(String devType,
     String pattern) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type.

pattern—pattern string being added.

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

getErrorPattern()

The method to retrieve the error pattern for the particular device type.

Usage:

```
Vector getErrorPattern(String devType) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type.

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

addIgnorePattern()

The method to add the pattern to be ignored for the particular device type.

Usage:

```
void addIgnorePattern(String devType,
     String pattern) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type

pattern—pattern string being added

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

deleteIgnorePattern()

The method to delete the pattern to be ignored for the particular device type.

Usage:

```
void deleteIgnorePattern(String devType,
```

```
 String pattern) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type.

pattern—pattern string being added.

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

getIgnorePattern()

The method to retrieve the pattern to be ignored for the particular device type.

Usage:

```
Vector getIgnorePattern(String devType) throws OperationFailedException,
    InvalidParameterException,
    NetworkException;
```

Parameters

devType—device type.

Exceptions

OperationFailedException—if the operation failed

InvalidParameterException—if a parameter is invalid

NetworkException—network exception

listAllDevices()

A public method which lists all devices's device IDs.

Usage:

```
Vector listAllDevices() throws NetworkException;
```

Returns

the vector containing a list of device-ids

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

listAllDevices()

A public method which lists the device-ids of those devices with given gateway-id.

Usage:

```
Vector listAllDevices(String gatewayID) throws NetworkException,
    InvalidParameterException;
```

Parameters

gatewayID—the gateway ID.

Returns

The vector containing a list of device IDs.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

InvalidParameterException—if given gateway id is invalid(eg: null string)

listAllDevices()

A public method which lists the device-ids of those devices with given gateway-id and device-type.

Usage:

```
Vector listAllDevices(String gatewayID,
     String deviceType) throws NetworkException,
    InvalidParameterException;
```

Parameters

gatewayID—the gateway ID.

deviceType—the device type.

Returns

The vector containing a list of device IDs.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

InvalidParameterException—if given gateway id or device type is invalid(eg: null string)

listAllDevices()

A public method which lists the device-ids of those devices with given gateway-id and emulated agent.

Usage:

```
Vector listAllDevices(String gatewayID,
     int emulatedAgent) throws NetworkException,
    InvalidParameterException;
```

Parameters

gatewayID—the gateway ID.

emulatedAgent—the agent type.

Returns

the vector containing a list of device-ids.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

InvalidParameterException—if given gateway id or device type is invalid(eg: null string)

createDevice()

A public method which create device in deviceinfo database.

Usage:

```
void createDevice(String devID,
     String gatewayID,
     String deviceType) throws DeviceAlreadyExistsException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device id

gatewayID—the gateway id.

deviceType—the device type.

Exceptions

DeviceAlreadyExistsException—if the specified device already exists

NetworkException—if sendRequest fails or the requested operation fails on server side

OperationFailedException—if the specified device already exists

InvalidParameterException—if given gateway id or device id is invalid(eg: null string)

createDevice()

A public method which create device in deviceinfo database.

Usage:

```
void createDevice(String devID,
     String gatewayID,
     String deviceType,
     int[] emulatedAgents) throws DeviceAlreadyExistsException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

gatewayID—the gateway ID.

deviceType—the device type.

emulatedAgents—the agents to emulate for the device.

Exceptions

DeviceAlreadyExistsException—if the specified device already exists

NetworkException—if sendRequest fails or the requested operation fails on server side

OperationFailedException—if the specified device already exists

InvalidParameterException—if given gateway id or device id is invalid(eg: null string)

getDeviceType()

A public method which gets device type of a given deviceID from deviceinfo database.

Usage:

```
String getDeviceType(String devID) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

Returns

A string representing the device type

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidParameterException—if given device id is invalid(eg: null string)

addDeviceHop()

A public method which adds a hop to specified device in deviceinfo database.

Usage:

```
void addDeviceHop(String devID,
     HopInfo hop) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    InvalidHopException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

hop—the HopInfo object to be added into deviceinfo database.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidParameterException—if given device id is invalid(eg: null string)

InvalidHopException—if the HopInfo object to be added is invalid(eg: HopInfo.devType is null)

getDeviceHop()

A public method which gets hop information from deviceinfo database.

Usage:

```
HopInfo getDeviceHop(String devID,
     int hopCnt) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    NoSuchHopException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

hopCnt—the index of the HopInfo object to be retrieved.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidParameterException—if given device id is invalid(eg: null string)

NoSuchHopException—if hopCnt exceeds the length of the HopInfo Vector or less than 0

deleteDevice()

A public method which deletes specified device from deviceinfo database.

Usage:

```
void deleteDevice(String devID) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if the specified device can't be deleted

InvalidParameterException—if given device id is invalid(eg: null string)

modifyDeviceHop()

A public method which updates the hop information of a specified device from deviceinfo database.

Usage:

```
void modifyDeviceHop(String devID,
     HopInfo[] hops) throws NetworkException,
    InvalidParameterException,
    DeviceNotFoundException,
    OperationFailedException,
    InvalidHopException;
```

Parameters

devID—the device ID.

hops—array of hop information.

Exceptions

InvalidParameterException—if the given parameter is invalid (e.g device-id is null)

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidHopException—if given hop information is invalid(eg: HopInfo.devType is null)

modifyDeviceHop()

A public method which updates the hop information of a specified device from deviceinfo database.

Usage:

```
void modifyDeviceHop(String devID,
     int[] indices,
     HopInfo[] hops) throws NetworkException,
    InvalidParameterException,
    DeviceNotFoundException,
    OperationFailedException,
    InvalidHopException;
```

Parameters

devID—the device ID.

hop—array of Hop Information.

hopnum—array of Hop Indices.

Exceptions

InvalidParameterException—if the given parameter is invalid (e.g device-id is null)

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidHopException—if given hop information is invalid(eg: HopInfo.devType is null)

getGatewayId()

A public method which returns the gateway id associated with a given deviceID.

Usage:

```
String getGatewayId(String devID) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

Returns

A string representing the gateway id

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidParameterException—if given device id is invalid(eg: null string)

getSimulatedAgents()

A public method which returns the set of agents that are simulated for a device with the given device ID.

Usage:

```
int getSimulatedAgents(String devID) throws DeviceNotFoundException,
    OperationFailedException,
    NetworkException,
    InvalidParameterException;
```

Parameters

devID—the device ID.

Returns

An array of integers that contains the set of simulated agents

Exceptions

NetworkException—if sendRequest fails or the requested operation fails on server side

DeviceNotFoundException—if the specified device does not exists

OperationFailedException—if there is error in this operation

InvalidParameterException—if given device id is invalid(eg: null string)

HopInfo API

package com.cisco.cns.imgw

public class HopInfo

A class which stores the necessary information of how to access a hop of some device.

Public Methods

HopInfo()

A public constructor of HopInfo with no parameter.

Usage:

```
HopInfo();
```

HopInfo()

A public constructor of HopInfo with parameters.

Usage:

```
HopInfo(String hopType,
     String ipAddr,
     int port,
     String user,
     String passwd) throws InvalidParameterException;
```

Parameters

hopType—the hop type

ipAddr—IP address

port—port number

user—username

passwd—password

Exceptions

InvalidParameterException— If parameters contain special character.

setHopType()

A public method which set the value of _hopType.

Usage:

```
void setHopType(String hopType) throws InvalidParameterException;
```

Parameters

hopType—the device type to be set.

Exceptions

InvalidParameterException— If parameter contains special character.

setIPAddr()

A public method which set the value of _ipAddr.

Usage:

```
void setIPAddr(String ipAddr) throws InvalidParameterException;
```

Parameters

ipAddr—the IP address to be set.

Exceptions

InvalidParameterException— If parameter contains special character.

setPort()

A public method which set the value of _port.

Usage:

```
void setPort(int port);
```

Parameters

port—the port number to be set.

setUserName()

A public method which set the value of _user.

Usage:

```
void setUserName(String user) throws InvalidParameterException;
```

Parameters

user—the username to be set.

Exceptions

InvalidParameterException— If parameter contains special character.

setPassword()

A public method which set the value of _passwd.

Usage:

```
void setPassword(String passwd) throws InvalidParameterException;
```

Parameters

passwd—the password to be set.

Exceptions

InvalidParameterException— If parameter contains special character.

getHopType()

A public method which get the value of _hopType.

Usage:

```
String getHopType();
```

Returns

device type.

getIPAddr()

A public method which get the value of _ipAddr.

Usage:

```
String getIPAddr();
```

Returns

IP address.

getPort()

A public method which get the value of _port.

Usage:

```
int getPort();
```

Returns

port number.

getUserName()

A public method which get the value of _user.

Usage:

```
String getUserName();
```

Returns

username.

getPassword()

A public method which get the value of _passwd.

Usage:

```
String getPassword();
```

Returns

password.

Exceptions

package com.cisco.cns.imgw

public class OperationFailedException extends Exception

A class which extends Exception class and will be thrown when addErrorPattern, deleteErrorPattern, addIgnorePattern, and deleteIgnorePattern fail

Public Methods

OperationFailedException()

A public constructor of OperationFailedException with no parameter.

Usage:

```
OperationFailedException();
```

OperationFailedException()

A public constructor of OperationFailedException with one parameter.

Usage:

```
OperationFailedException(String errMsg);
```

Parameters

errMsg—the error message

public class DeviceNotFoundException extends Exception

A class which extends Exception class and will be thrown if the specified device can not be found where it should be.

DeviceNotFoundException()

A public constructor of DeviceNotFoundException with no parameter.

Usage:

```
DeviceNotFoundException();
```

DeviceNotFoundException()

A public constructor of DeviceNotFoundException with one parameter.

Usage:

```
DeviceNotFoundException(String errCode,
     String errMsg);
```

Parameters

errCode—the error code

errMsg—the error message

public class InvalidHopException extends InvalidParameterException

A class which extends InvalidParameterException class and will be thrown if devType in HopInfo is null string.

InvalidHopException()

A public constructor of InvalidHopException with no parameter.

Usage:

```
InvalidHopException();
```

InvalidHopException()

A public constructor of InvalidHopException with one parameter.

Usage:

```
InvalidHopException(String errMsg);
```

Parameters

errMsg—the error message

public class InvalidParameterException extends Exception

A class which extends Exception class and will be thrown when gateway-id/device-id is null or InvalidHopInfoException occurs.

InvalidParameterException()

A public constructor of InvalidParameterException with no parameter.

Usage:

```
InvalidParameterException();
```

InvalidParameterException()

A public constructor of InvalidParameterException with one parameter.

Usage:

```
InvalidParameterException(String errMsg);
```

Parameters

errMsg—the error message

public class NetworkException extends Exception

A class which extends Exception class and will be thrown when sendRequest fails

NetworkException()

A public constructor of NetworkException with no parameter.

Usage:

```
NetworkException();
```

NetworkException()

A public constructor of NetworkException with one parameter.

Usage:

```
NetworkException(String errMsg);
```

Parameters

errMsg--the error message

public class NoSuchHopException extends Exception

A class which extends Exception class and will be thrown if the hop to be looked up does not exist.

NoSuchHopException()

A public constructor of NoSuchHopException with no parameter.

Usage:

```
NoSuchHopException();
```

NoSuchHopException()

A public constructor of NoSuchHopException with one parameter.

Usage:

```
NoSuchHopException(String errMsg);
```

Parameters

errMsg—the error message

public class DeviceAlreadyExistsException extends Exception

A class which extends Exception class and will be thrown if the specified device already exists where it should not.

DeviceAlreadyExistsException()

A public constructor of DeviceAlreadyExistsException with no parameter.

Usage:

```
DeviceAlreadyExistsException();
```

DeviceAlreadyExistsException()

A public constructor of DeviceAlreadyExistsException with one parameter.

Usage:

```
DeviceAlreadyExistsException(String errCode,
     String errMsg);
```

Parameters

errCode—the error code

errMsg—the error message

APPENDIX **A**

Code Samples

Using the Event API and the Namespace Mapper API

The code examples that follow illustrate the use of the Event API and the Namespace Mapper API. To use the examples, you need to have populated your LDAP directory with application namespace objects (see the section Object Model in Chapter 4, "Namespace Mapping Service") and you must have your host system configured in server mode (see *Cisco Configuration Engine Administration Guide*). You also need to have a correctly configured *nsm.conf* file with the right directive.

The file *nsm_client.h* is located in the */<INSTALLDIR>/CSCOesdk/include* directory. The Java package **com.cisco.cns.nsm.client** is contained in the file *NSMClient.jar*, which is located in the *<INSTALLDIR>/CSCOesdk/java* directory.

C++

CiscoApp.cpp — Invokes Listener and Sender objects

```
/*
 * Copyright (c) 2000, 2001, 2002, 2003 by Cisco Systems, Inc.
 * All rights reserved worldwide.
 *
 * Warning: This program is protected by copyright law and international
 * treaties.  Unauthorized redistribution of this program, or any portion
 * of it, may result in severe civil and criminal penalties, and will be
 * prosecuted to the maximum extend possible under law
 *
 *-------------------------------------------------------------------

 * CiscoApp.cpp: This sample program uses the CNS Event Services API.
 * It calls a listener object to listen for events and a sender object
 * to publish events.
 * Usage:
 *       CiscoApp [parameters]
 *
 *  Command-line parameters:
        -service service port           UDP port on which the
                                        rv daemon communicates with other
                                        daemons. Default. Default: 7500

        -network network interface      Network interface for outbound messages
                                        from the application. Not needed for
                                        hosts with 1 network interface
```

```
        -daemon  rvd listen port        The rv daemon listens for transport
                                        connections from applications on this
                                        TCP port. Default: 7500

        -file file name                 File from which to read XML
                                        configuration.
        -app application namespace      Namespace to which to attach to.
                                        Required for Namespace Mapper.
                                        Default: config

        -id identifier for device/      String to uniquely identify a device
                group                   or a group. If the user wants to
                                        publish to just one device this would
                                        be the device id. If the user wants to
                                        publish to a group of devices, this
                                        would be the group id


        -element 0 | 1                  Type of element that the above parameter
                                        "-id" identifies. If this a device
                                        the value should be 0. If this is a
                                        group, the value should be 1.

*/



#include <iostream.h>
#ifndefWIN32
#include <unistd.h>
#else
#include <windows.h>
#endif
#include "CiscoAppListen.h"
#include "CiscoAppSend.h"

#ifdef WIN32
#    define SLEEP Sleep
#else
#    define SLEEP sleep
#endif

/* The following three parameters specify the
    parameters for Tibco transport to the
    Tibco daemon
*/
static char *service = NULL;
static char *network = NULL;
static char *daemon_  = NULL;

/* File containing the XML payload */
static char *configFile = "config-file";

/* Unique Device Identifier */
static char *identifier = "D1";

/* Default application namespace */
static char *appName = "config";

/* Type of element passed to Namespace Mapper
   0 - DEVICE
   1 - GROUP
*/
```

```
static int element = 0;

/* The config agent on the device subscribes to cisco.mgmt.cns.config.load.
   So this application publishes on the same subject
*/
char *DEFAULT_PUB_SUBJECT = "cisco.mgmt.cns.config.load";

/* The config agent on the device, on receiving configuration
   from the application publishes on the following subjects,
   indicating success, failure and success with warning
   respectively. So this application subscribes to these
   events
*/
char *DEFAULT_SUB_SUBJECT_SUCCESS = "cisco.mgmt.cns.config.complete";
char *DEFAULT_SUB_SUBJECT_FAILURE = "cisco.mgmt.cns.config.failure";
char *DEFAULT_SUB_SUBJECT_WARNING = "cisco.mgmt.cns.config.warning";

tibrvTransport transport;

void
usage(void)
{
    fprintf(stderr,"CiscoApp [-service service] [-network network] \n");
    fprintf(stderr,"            [-daemon daemon] [-file filename] \n");
    fprintf(stderr,"            [-id identifier of device/group] [-element 0 or 1] \n");
    fprintf(stderr,"            [-app Application namespace] \n");
    exit(1);
}


/* get_InitParams: parses command-line parameters */
int
get_InitParms(
    int         argc,
    char*       argv[])
{

    int i = 1;

    while ( i+2 <= argc && *argv[i] == '-' )
    {

        if(strcmp(argv[i], "-service") == 0)
        {
            service = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-network") == 0)
        {
            network = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-daemon") == 0)
        {
            daemon_ = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-file") == 0)
        {
            configFile = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-app") == 0)
        {
```

```
            appName = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-id") == 0)
        {
            identifier = argv[i+1];
            i+=2;
        }
        else if(strcmp(argv[i], "-element") == 0)
    {
        element = atoi (argv[i+1]);
        if (element != NSMClient::DEVICE && element != NSMClient::GROUP)
        {
            printf ("Invalid input for element.\n");
            usage();
        }
        i += 2;
    }
        else
        {
            usage();
        }
    }

    return( i );
}

void init ()
{
        tibrv_status err;

        /*
         * Create internal TIB/Rendezvous machinery
         */
        err = tibrv_Open();
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Failed to open TIB/RV --%s\n",
                        tibrvStatus_GetText(err));
            exit(-1);
        }

    /* Create Transport to rv daemon */
    err = tibrvTransport_Create (&transport, NULL, NULL, NULL);
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Failed to open transport --%s\n",
                        tibrvStatus_GetText(err));
            exit(-1);
        }

}

void fini ()
{
    /* Destroy Transport to rv daemon */
    tibrvTransport_Destroy (transport);

        /*
         * tibrv_Close() will destroy the transport mechanism.
         */

        tibrv_Close();
```

```
}


int main (int argc, char **argv)
{
    // parse arguments for possible optional
        // parameters.
    get_InitParms (argc, argv);

    init ();

    // Create listeners for success, failure and warning messages
    CiscoAppListen listener(transport);
    listener.subscribe (DEFAULT_SUB_SUBJECT_SUCCESS, identifier, appName, element);
    listener.subscribe (DEFAULT_SUB_SUBJECT_FAILURE, identifier, appName, element);
    listener.subscribe (DEFAULT_SUB_SUBJECT_WARNING, identifier, appName, element);


        //  Create sender objects to publish to the device or group of devices
    CiscoAppSend sender(transport);
    sender.publish (DEFAULT_PUB_SUBJECT, configFile, identifier, appName, element);


    // Sleep because we would exit the program otherwise
       // and not receive published events
       // The other way to do it is to do tibrvQueue_Dispatch in the
       // same thread as main so that we would be in a loop
        while (1)
        SLEEP(100);
    fini ();
        return 0;
}
```

CiscoAppListen.cpp — Sample Listener

```
 * CiscoAppListen - sample Rendezvous message subscriber
 *
 * Copyright (c) 2000, 2001, 2002, 2003 by Cisco Systems, Inc.
 * All rights reserved worldwide.
 *
 * Warning: This program is protected by copyright law and international
 * treaties.  Unauthorized redistribution of this program, or any portion
 * of it, may result in severe civil and criminal penalties, and will be
 * prosecuted to the maximum extend possible under law
 *
 * This class subscribes events to a specified
 * subject.
 *
 * Optionally the user may specify communication parameters for
 * tibrvTransport_Create.
 *
 * If none are specified the following defaults are used:
 *  service     "rendezvous" or "7500/udp"
 *  network     the result of gethostname
 *  daemon      "tcp:7500"
 *
 */

#include "CiscoAppListen.h"

/* Constructor : initializes data members */
```

```
CiscoAppListen::CiscoAppListen (tibrvTransport &t) :sub_transport(t)
{
    identifier = NULL;
    tibrv_status err;

        err = tibrvQueue_Create(&queue) ;
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Failed to create queue --%s\n",
                        tibrvStatus_GetText(err));
            exit(-1);
        }

        err = tibrvDispatcher_Create(&dispatcher,queue) ;
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Failed to create dispatcher --%s\n",
                        tibrvStatus_GetText(err));
            exit(-1);
        }

}

/**
 *  This method calls the Namespace Mapper to obtain mapping for the
 *  given subject , and subscribes to the mapped subject
 *
 *  @param sub specifies the subject on which the application wishes to
 *             publish. This will be mapped using the NSM API calls
 *                  XML payload
 *  @param id specifies the identifier of the group/device
 *  @param app specifies the namespace for Namespace mapping
 *  @param elem specifies whether the identifier refers to a group or a
 *              device
 */
int CiscoAppListen::subscribe (char *sub, char *id, char *app, int elem)
{

        // Set the Data members based on input parameters
        setDataMembers (id) ;

        // Obtain mapping for original subject
    NSMResult *result = getMapping (app, id, sub, NSMClient::SUBSCRIBE, elem );

    if (result != NULL)
    {
        char *tmp;
        for (NSMResultIterator iter(*result); !iter.isEnd(); iter.advance())
        {
            tmp = iter.value();
            printf ("Mapped subject for listening.\n ");

        // subscribe to every mapped subject
        subscribeMessage (tmp);
            }

            delete result;
    }
    return 0;
}



/* This private method sets the private data memebers of this class */
```

```
void CiscoAppListen::setDataMembers (char *id)
{
    if (id != NULL)
    {
        identifier = strdup (id);
    }
}

/* This private method actually creates a listener for the mapped subject */
void CiscoAppListen::subscribeMessage (char *subjectStr)
{
    tibrv_status err;
    tibrvEvent listenId;

    // Create a tibco listener
    err =  tibrvEvent_CreateListener(&listenId, queue,
                                     my_callback, sub_transport,
                                     subjectStr, identifier);

        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Error %s listening to \"%s\"\n",
                     tibrvStatus_GetText(err), subjectStr);
            exit(-1);
        }
    printf ("Subscribing to %s.\n", subjectStr);
}

/**
 * This private method contacts the Namespace Mapper and obtains a
 * namespace specific mapping for the subject
 * @param appName specifies the application namespace
 * @param id specifies the identifier of the device or group
 * @param subject specifies the subject to map
 * @param action specifies the operation : PUBLISH or SUBSCRIBE
 * @param element specifies the type that id refers to - GROUP or DEVICE
 */
NSMResult *CiscoAppListen::getMapping (char *app, char *id, char *subjectStr, int action,
int elem)
{
    char *tmp=NULL;
    int rc;


    /*** Creating the nsm client ***/
    NSMClient *nsmc = NULL;
    nsmc = new NSMClient ("./nsm.conf");


    /*** Calling the attach method with an application name ***/
    rc = nsmc->attach(app);

    /*** If we can't attach successfully, exit the program ***/
    if(NSMClient::SUCCESS != rc)
    {
        printf ("attach failure.\n");
        exit(-1);
    }

    NSMResult *result = new NSMResult();

    NSMResolveRequest request(id, subjectStr, action, elem);

    /*** Calling the resolve method ***/
```

```
    rc = nsmc->resolve(*result, request);

    /*** Detach from the client ***/
    nsmc->detach();

    delete nsmc ;

    /*** if resolve is successful, return the results ***/
    if(NSMClient::SUCCESS == rc)
    {
        printf ("Resolve successful.\n");
    return result;
    }
    else
    {
        printf ("Resolve failure.\n" );
    return NULL;
    }

}

CiscoAppListen::~CiscoAppListen ()
{
    if (identifier != NULL)
        delete [] identifier;
        /*
        * Destroy dispatcher
         */
        tibrvDispatcher_Destroy (dispatcher);

        /*
        * Destroy queue
         */
        tibrvQueue_Destroy (queue);

}

void my_callback(
    tibrvEvent          event,
    tibrvMsg            message,
    void*               closure)
{
    time_t              now;
    struct tm*          tmnow;

    const char*         send_subject  = NULL;
    const char*         reply_subject = NULL;
    const char*         theString     = NULL;

    /*
     * Get the subject name to which this message was sent.
     */
    tibrvMsg_GetSendSubject(message, &send_subject);

    /*
     * If there was a reply subject, get it.
     */
    tibrvMsg_GetReplySubject(message, &reply_subject);
    /*
     * Convert the incoming message to a string.
     */
    tibrvMsg_ConvertToString(message, &theString);

    /*
```

```
     * Get the local time information.
     */
    (void) time(&now);
    tmnow = localtime(&now);

    /*
     * Print it all out.
     */
    printf("[%04d-%02d-%02d %02d:%02d:%02d]: ",
                tmnow->tm_year+1900, tmnow->tm_mon + 1, tmnow->tm_mday,
                tmnow->tm_hour, tmnow->tm_min, tmnow->tm_sec);
    if (reply_subject)
        printf("subject=%s, reply=%s, message=%s\n",
               send_subject, reply_subject, theString);
    else
        printf("subject=%s, message=%s\n",
               send_subject, theString);

    fflush(stdout);
}
```

CiscoAppSend.cpp — Sample Sender

```
/*
 * CiscoAppSend - sample Rendezvous message publisher
 *
 * Copyright (c) 2000, 2001, 2002, 2003 by Cisco Systems, Inc.
 * All rights reserved worldwide.
 *
 * Warning: This program is protected by copyright law and international
 * treaties.  Unauthorized redistribution of this program, or any portion
 * of it, may result in severe civil and criminal penalties, and will be
 * prosecuted to the maximum extend possible under law
 *
 * This class publishes events on a specified subject.
 * A field named "DATA" will be created to hold the string in each
 * message.
 *
 * Optionally the user may specify communication parameters for
 * tibrvTransport_Create.
 *
 * If none are specified the following defaults are used:
 *  service     "rendezvous" or "7500/udp"
 *  network     the result of gethostname
 *  daemon      "tcp:7500"
 *
 */



#include "CiscoAppSend.h"

/* Constructor : initializes data members */
CiscoAppSend::CiscoAppSend (tibrvTransport &t) : pub_transport(t)
{
    payload = NULL;
    fileName = NULL;
}

/**
*  This method reads a payload string from a file, calls the namespace
*  mapper to map the original subject to a namespace specific subject,
```

```
*  and then publishes the payload on the mapped subject.
*
*  @param sub specifies the subject on which the application wishes to
*             publish. This will be mapped using the NSM API calls
*  @param filename specifies the name of the file from which to read the
*                  XML payload
*  @param id specifies the identifier of the group/device
*  @param app specifies the application namespace to use while mapping
*  @param elem specifies whether the identifier refers to a group or a
*              device
*/

int CiscoAppSend::publish (char *sub, char *filename, char *id, char *app, int elem)
{
        // Set the Data members based on input parameters
        setDataMembers (filename) ;

        // Read config from input file
        readConfigFromFile ();
        if (payload == NULL)
        {
            printf("payload is NULL\n");
        }

    NSMResult *result = getMapping (app, id, sub, NSMClient::PUBLISH, elem);

    if (result != NULL)
    {
        char *tmp = NULL;

        for (NSMResultIterator iter(*result); !iter.isEnd(); iter.advance())
            {
        tmp = iter.value();

            printf ("Mapped subject for publishing %s\n", tmp );

        publishMessage (tmp, payload);
            }

        delete result;
    }
    return 0;
}


/* This private method reads XML data from a file */
void CiscoAppSend::readConfigFromFile ()
{
    FILE *fptr = NULL;

        if(fptr = fopen(fileName, "r"))
        {
            if (0 != fseek(fptr, 0L, SEEK_END))
            {
                printf("fseek to SEEK_END failed!\n");
                exit(0);
            }

            int file_length = ftell(fptr);

            rewind(fptr);

            if (NULL == (payload = (char *) malloc(file_length + 1)))
            {
```

```
                printf("malloc failed!\n");
                exit(1);
            }

        int c,i;

            for(i = 0; (c = fgetc(fptr))!= EOF; ++i)
            {
                payload[i] = (unsigned char)c;
            }
            payload[i] = '\0';
        printf ("XML String is %s\n", payload);

        fclose (fptr);
        }

}

/* This private method sets the private data memebers of this class */
void CiscoAppSend::setDataMembers (char *filename)
{
    if (filename != NULL)
    {
        fileName = strdup (filename);
    }
}

/* This private method actually publishes the mapped subject on the bus
        with the given payload */
void CiscoAppSend::publishMessage (char *subjectStr, char *payloadStr)
{
    tibrvMsg message;
        tibrv_status err;

        /*
         * Create message
         */
        err = tibrvMsg_Create(&message);
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "Failed to create message --%s\n",
                tibrvStatus_GetText(err));
            exit(-1);
        }

        /* Set the subject name */
        err = tibrvMsg_SetSendSubject(message, subjectStr);
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "%s in setting subject \"%s\" to \"%s\"\n",
                    tibrvStatus_GetText(err),
                    subjectStr);
        exit(-1);
        }

        /* Update data portion of the message */
        err= tibrvMsg_UpdateOpaque(message, FIELD_NAME, payloadStr, strlen(payloadStr));
        if (err != TIBRV_OK)
        {
            fprintf(stderr, "%s in setting data \n",
                    tibrvStatus_GetText(err)
                    );
        exit(-1);
        }
```

```
    /* Publish the message over the transport */
    err = tibrvTransport_Send(pub_transport, message);

        if ( err != TIBRV_OK)
        {
            fprintf(stderr, "%s in sending \"%s\" \n",
                    tibrvStatus_GetText(err),
                    subjectStr);
        exit(-1);
        }

    printf ("Published on %s.\n", subjectStr);

        /*
         * Destroy message
        */
        tibrvMsg_Destroy(message);

}

/**
 * This private method contacts the Namespace Mapper and obtains a
 * namespace specific mapping for the subject
 * @param app specifies the application namespace
 * @param id specifies the identifier of the device or group
 * @param subjectStr specifies the subject to map
 * @param action specifies the operation : PUBLISH or SUBSCRIBE
 * @param elem specifies the type that id refers to - GROUP or DEVICE
 */
NSMResult* CiscoAppSend::getMapping (char *app, char *id, char *subjectStr, int action,
int elem)
{
    char *tmp=NULL;
    int rc;

    /*** Creating the nsm client ***/
    NSMClient *nsmc = NULL;
    nsmc = new NSMClient("./nsm.conf");

    /*** Calling the attach method with an application name ***/
    rc = nsmc->attach(app);

    /*** If we can't attach successfully, exit the program ***/
    if(NSMClient::SUCCESS != rc)
    {
        printf ("attach failure.\n");
        exit(-1);
    }
    NSMResult *result = new NSMResult();
    NSMResolveRequest request(id, subjectStr, action, elem);

    /*** Calling the resolve method ***/
    rc = nsmc->resolve(*result, request);

    /*** Detach from the client ***/
    nsmc->detach();

    delete nsmc ;

    /*** if resolve is successful, return the results ***/
    if(NSMClient::SUCCESS == rc)
    {
        printf ("Resolve successful.\n");
```

```
    return result;
    }
    else
    {
        printf ("Resolve failure.\n");
    return NULL;
    }

}

CiscoAppSend::~CiscoAppSend ()
{
    if (payload != NULL)
        delete [] payload;

    if (fileName != NULL)
        delete [] fileName;

}
```

CiscoAppListen.h

```
/**
 * CiscoAppSend.h: Defines a class that acts as a rendezvous subscriber
 *
 * Copyright (c) 2000, 2001, 2002, 2003 by Cisco Systems, Inc.
 * All rights reserved worldwide.
 *
 * Warning: This program is protected by copyright law and international
 * treaties.  Unauthorized redistribution of this program, or any portion
 * of it, may result in severe civil and criminal penalties, and will be
 * prosecuted to the maximum extend possible under law
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "nsm_client.h"
#include "tibrv/tibrv.h"

void my_callback(
    tibrvEvent          event,
    tibrvMsg            message,
    void*               closure);

class CiscoAppListen
{
public:
    CiscoAppListen (tibrvTransport &t);

    int subscribe (char *sub, char *id, char *app, int elem);

    ~CiscoAppListen ();
private:

    void setDataMembers (char *id);

    void subscribeMessage (char *subject);

    NSMResult *getMapping (char *appName, char *id, char *subject, int action, int
element);
```

```
    char *identifier;
    tibrvTransport sub_transport;
    tibrvQueue queue;
    tibrvDispatcher dispatcher;
};
```

CiscoAppSend.h

```
/**
 * CiscoAppSend.h: Defines a class that acts as a rendezvous publisher
 *
 * Copyright (c) 2000, 2001, 2002, 2003 by Cisco Systems, Inc.
 * All rights reserved worldwide.
 *
 * Warning: This program is protected by copyright law and international
 * treaties.  Unauthorized redistribution of this program, or any portion
 * of it, may result in severe civil and criminal penalties, and will be
 * prosecuted to the maximum extend possible under law
 */

#include <stdlib.h>
#include <string.h>
#include "nsm_client.h"
#include "tibrv/tibrv.h"

const int BUFLEN = 25;

#define FIELD_NAME  "DATA"

class CiscoAppSend
{
public:
    CiscoAppSend (tibrvTransport &t);

    int publish (char *sub, char *filename, char *id, char *app, int elem);

    ~CiscoAppSend ();
private:

    void readConfigFromFile ();

    void setDataMembers (char *filename);

    void publishMessage (char *subject, char *payload);

    NSMResult *getMapping (char *appName, char *id, char *subject, int action, int
element);


    char *payload;
    char *fileName;

    tibrvTransport pub_transport;
};
```

Makefiles

Sample Makefile for CC 6.0

```
# *******************************************************
#
# To compile the examples, please set the necessary items for your environment
# then make!
#
#########################################################
#
O=.o
EXE=
RM=rm -f
OUT=-o
INST_DIR=/opt/CSCOesdk
#
CC = CC
#OPT_DBG_FLAGS = -g
LDFLAGS=-L$(INST_DIR)/lib
INCFILES=-I$(INST_DIR)/include
#CFLAGS=$(OPT_DBG_FLGS) $(INCFILES)
EVENT_LIBS=-ltibrv
NSM_LIBS=-lHttpWrapper -lACE -lnsmclient -lxerces-c2_5
#
#
# if you have a System V Release 4 system you need these libs as well
#
SOCKET_LIBS=-lsocket -lgen -lnsl
#
#
# Specify a thread library. This is required on most unixes for things
# to have things work correctly. On Solaris use -lthreads
#
TLIBS=-lthread
#

LIBS=  $(EVENT_LIBS) $(NSM_LIBS) $(SOCKET_LIBS) $(SYS_LIBS) $(TLIBS)


all:CiscoApp$(EXE)

clean:
    $(RM) *$(O)
    $(RM) core
    $(RM) CiscoApp$(EXE)

CiscoApp$(EXE):CiscoAppSend$(O) CiscoAppListen$(O) CiscoApp$(O)
    $(CC) $(LDFLAGS) CiscoApp$(O) CiscoAppSend$(O) CiscoAppListen$(O) $(LIBS) $(OUT) $@

CiscoAppSend$(O): CiscoAppSend.cpp
    $(CC) CiscoAppSend.cpp $(INCFILES) -c $(OUT) $@

CiscoAppListen$(O): CiscoAppListen.cpp
    $(CC) CiscoAppListen.cpp $(INCFILES) -c $(OUT) $@

CiscoApp$(O): CiscoApp.cpp
    $(CC) CiscoApp.cpp $(INCFILES) -c $(OUT) $@
```

Java

CiscoApp.java — Invokes Listener and Sender objects

```
/*
* Copyright (c) 2001, 2002, 2003 by cisco Systems, Inc.
*
* CiscoApp.java: This sample program uses the CNS Event Services API.
* It calls a listener object to listen for events and a sender object
* to publish events.
*  Usage:
*   java CiscoApp [parameters]
*
*  Command-line parameters:
    -service service port         UDP port on which the
                   rv daemon communicates with other
                   daemons. Default. Default: 7500

    -network network interface Network interface for outbound messages
                   from the application. Not needed for
                   hosts with 1 network interface

    -daemon  rvd listen portThe rv daemon listens for transport
                   connections from applications on this
                   TCP port. Default: 7500

    -file file nameFile from which to read XML
                   configuration.

    -app application namespaceNamespace to which to attach to.
                   Required for Namespace Mapper.
                   Default: config

    -id identifier for device/String to uniquely identify a device
        group          or a group. If the user wants to
                   publish to just one device this would
                   be the device id. If the user wants to
                   publish to a group of devices, this
                   would be the group id


    -element 0 | 1Type of element that the above parameter"-id" identifies. If this a
device
                   the value should be 0. If this is a
                   group, the value should be 1.

*/

import com.cisco.cns.nsm.client.*;

public class CiscoApp
{
    /* The following three parameters specify the
       parameters for Tibco transport to the
       Tibco daemon
    */
    static String service = null;
    static String network = null;
    static String daemon  = null;

    /* File containing the XML payload */
    static String configFile = "config-file";
```

```
 /* Unique Device Identifier */
 static String identifier = "D1";

/* Default application namespace */
 static String appName = "config";

/* Type of element passed to Namespace Mapper
   0 - DEVICE
   1 - GROUP
*/
 static int element = 0;

 /* The config agent on the device subscribes to cisco.mgmt.cns.config.load.
    So this application publishes on the same subject
 */
 static String DEFAULT_PUB_SUBJECT = "cisco.mgmt.cns.config.load";

 /* The config agent on the device, on receiving configuration
    from the application publishes on the following subjects,
    indicating success, failure and success with warning
    respectively. So this application subscribes to these
    events
 */
 static String DEFAULT_SUB_SUBJECT_SUCCESS = "cisco.mgmt.cns.config.complete";
 static String DEFAULT_SUB_SUBJECT_FAILURE = "cisco.mgmt.cns.config.failure";
 static String DEFAULT_SUB_SUBJECT_WARNING = "cisco.mgmt.cns.config.warning";


 /* get_InitParams: parses command-line parameters */
 static int get_InitParams(String[] args)
 {
     int i=0;
     while(i < args.length-1 && args[i].startsWith("-"))
     {
         if (args[i].equals("-service"))
         {
             service = args[i+1];
             i += 2;
         }
         else
         if (args[i].equals("-network"))
         {
             network = args[i+1];
             i += 2;
         }
         else
         if (args[i].equals("-daemon"))
         {
             daemon = args[i+1];
             i += 2;
         }
     else
         if (args[i].equals("-file"))
         {
             configFile = args[i+1];
             i += 2;
         }
     else
         if (args[i].equals("-app"))
         {
             appName = args[i+1];
             i += 2;
         }
```

```
        else
            if (args[i].equals("-id"))
            {
                identifier = args[i+1];
                i += 2;
            }
            else
            if (args[i].equals("-element"))
            {
                element = Integer.parseInt (args[i+1]);
                if (element != NSMClient.DEVICE && element != NSMClient.GROUP)
                {
                        System.out.println ("Not a valid element type: " + element);
                        usage ();
                }
                i += 2;
            }
        else
        usage();
        }
        return i;
    }


    // print usage information and quit
    static void usage()
    {
        System.err.println("Usage: java CiscoApp [-service service] [-network network]");
        System.err.println("            [-daemon daemon] [-file filename] [-id identifier
of device/group] [-element 0 or 1] [-app Application namespace]");
        System.exit(0);
    }


    public static void main (String[] args)
    {
        // parse arguments for possible optional
        // parameters.
        int i = CiscoApp.get_InitParams(args);

    // Create listeners for success, failure and warning messages
    CiscoAppListen listener = new CiscoAppListen();
    listener.init (service, network, daemon);
    listener.subscribe (DEFAULT_SUB_SUBJECT_SUCCESS, identifier, appName, element);
    listener.subscribe (DEFAULT_SUB_SUBJECT_FAILURE, identifier, appName, element);
    listener.subscribe (DEFAULT_SUB_SUBJECT_WARNING, identifier, appName, element);


    //  Create sender objects to publish to the device or group of devices
    CiscoAppSend sender = new CiscoAppSend();
    sender.init(service, network, daemon);
    sender.publish(DEFAULT_PUB_SUBJECT, configFile, identifier, appName, element );
    sender.fini();

    }
}
```

CiscoAppListen.java — Sample Listener

```
/*
```

```
 * Copyright (c) 2001, 2002, 2003 by cisco Systems, Inc.
 *
 * CiscoAppListen - sample Rendezvous message subscriber
 *
 * This class subscribes events to a specified
 * subject.
 *
 * Optionally the user may specify communication parameters for
 * tibrvTransport_Create.
 *
 * If none are specified the following defaults are used:
 *  service     "rendezvous" or "7500/udp"
 *  network     the result of gethostname
 *  daemon      "tcp:7500"
 *
 */

import java.util.*;
import com.tibco.tibrv.*;
import com.cisco.cns.nsm.client.*;

public class CiscoAppListen implements TibrvMsgCallback
{
    String subject = "cisco.mgmt.cns.config.failure";
    String service = null;
    String network = null;
    String daemon  = null;

    String identifier = "DeviceId1";
    String appName = "config";
    int element = NSMClient.DEVICE;

    String FIELD_NAME = "DATA";

    boolean init=false;

    // Transport to rendezvous daemon
    TibrvTransport transport = null;
    TibrvQueue queue = null;
    TibrvDispatcher dispatcher = null;

    /**
     *  This method calls the Namespace Mapper to obtain mapping for the
     *  given subject , and subscribes to the mapped subject
     *
     *  @param sub specifies the subject on which the application wishes to
     *             publish. This will be mapped using the NSM API calls
     *                  XML payload
     *  @param id specifies the identifier of the group/device
     *  @param app specifies the namespace for Namespace mapping
     *  @param elem specifies whether the identifier refers to a group or a
     *              device
     */
    public int subscribe(String sub, String id, String app, int elem)
    {
    // Make sure the Tibco event mechanism is initialized
        if (!init)
    {
        System.out.println ("Not initialized. Call init method first. " );
        return -1;
    }
System.out.println ("In listen" );
    // Set the Data members based on input parameters
        setDataMembers (sub, id,  app, elem) ;
```

```
System.out.println ("After setData" );

        setupQueue ();

    // Obtain mapping for subject from Namespace Mapper
    NSMResult nsmMapping = getMapping (appName, identifier, subject, NSMClient.SUBSCRIBE,
element );
    if (nsmMapping != null)
    {

        // If mapping succeeded, publish on each of the mapped
        //  subjects

            NSMResultIterator iter = null;
        for (iter = new NSMResultIterator(nsmMapping); !iter.isEnd(); iter.advance())
        {
        String mappedSubject = iter.value();
            System.out.println("Mapped subject for publishing " + mappedSubject);
        subscribeMessage (mappedSubject);
            }

    }
    return 0;
    }

    /**
     * This callback method will be called when an event is received on the
     * subscribed subject.
     */
    public void onMsg(TibrvListener listener, TibrvMsg msg)
    {
            System.out.println((new Date()).toString()+
                            ": subject="+msg.getSendSubject()+
                            ", message="+msg.toString()+
                " from device " + listener.getClosure().toString() );
        System.out.flush();
    }


    /* This private method sets the private data memebers of this class */
    void setDataMembers (String sub, String id, String app, int elem)
    {
        if (sub != null)
                subject = sub;
        if (id != null)
                identifier = id;
        if (app != null)
                appName = app;
        if (elem != 0)
                element = elem;

    }

    /* This private method creates a queue in which events will be queued
     * A dispatcher thread will dispatch the events to the application.
     * The callback method runs in the dispatcher thread
     */
    void setupQueue ()
    {
    try
    {
        queue = new TibrvQueue();
        dispatcher = new TibrvDispatcher("Dispatcher",queue);
    }
```

```
    catch (TibrvException e) {
        System.err.println("Failed to create queue:");
        e.printStackTrace();
        System.exit(-1);
    }
}

/* This private method actually creates a listener for the mapped subject */
void subscribeMessage (String subject)
{
    try
    {
           new TibrvListener(queue,
                        this,transport,subject,identifier);
            System.err.println("Listening on: "+subject);
    }
    catch (TibrvException e) {
        System.err.println("Failed to create listener:");
        e.printStackTrace();
        System.exit(-1);
    }

}

/**
 * This method initializes the Tibco mechanism and creates a transport
 * between the application and the rendezvous daemon
 * @param service specifies the UDP port for rendezvous communication
 * @param network specifies the network interface for outbound messages
 * @param daemon specifies the TCP port on which the daemon listens for
 *               connections from applications
 */
void init (String sevice, String network, String daemon)
{

    // open Tibrv in native implementation
    try
    {
        Tibrv.open(Tibrv.IMPL_NATIVE);
    }
    catch (TibrvException e)
    {
        System.err.println("Failed to open Tibrv in native implementation:");
        e.printStackTrace();
        System.exit(-1);
    }

    try
    {
        transport = new TibrvRvdTransport(service,network,daemon);
    }
    catch (TibrvException e)
    {
        System.err.println("Failed to create TibrvRvdTransport:");
        e.printStackTrace();
        System.exit(-1);
    }

init = true;
}

 /**
 * This method closes the transport and closes the Tibco event mechanism
 */
```

```
    void fini ()
    {

        // Close Tibrv, it will cleanup all underlying memory, destroy
        // transport and guarantee delivery.
        try
        {
            Tibrv.close();
        }
        catch(TibrvException e)
        {
            System.err.println("Exception dispatching default queue:");
            e.printStackTrace();
            System.exit(-1);
        }
    init=false;
    }


    /**
     * This private method contacts the Namespace Mapper and obtains a
     * namespace specific mapping for the subject
     * @param appName specifies the application namespace
     * @param id specifies the identifier of the device or group
     * @param subject specifies the subject to map
     * @param action specifies the operation : PUBLISH or SUBSCRIBE
     * @param element specifies the type that id refers to - GROUP or DEVICE
     */
    NSMResult getMapping (String appName, String id, String subject, int action, int
element)
    {
        int rc;
        NSMResult result = new NSMResult();

        //Creating a NSMClient instance
        NSMClient nsmc = NSMClient.create();

        //Attaching to an application called App1
        rc = nsmc.attach(appName);

        //Testing for the status of attach
        if(NSMClient.SUCCESS != rc)
        {
            System.out.println("we got an attach error");
            System.exit(-1);
        }

System.out.println ("Attached to " + appName + ". Calling resolve" );
        //Calling the resolve method with parameters obtained above

        NSMResolveRequest request = new NSMResolveRequest(id, subject,
        action, element);

        rc = nsmc.resolve(result, request);
System.out.println ("After resolve" );

        //detaching the connection
        nsmc.detach();


        //Check to see the an empty vector gets returned.
        if(NSMClient.SUCCESS == rc)
        {
            System.out.println("Obtained mapping.");
```

```
        return result;
        }
        else
        {
            System.out.println("no result");
        return null;
        }

    }

}
```

CiscoAppSend.java — Sample Sender

```
/*
 * Copyright (c) 2001, 2002, 2003 by cisco Systems, Inc.
 *
 * CiscoAppSend - sample Rendezvous message publisher
 *
 * This class publishes events on a specified
 * subject.
 * A field named "DATA" will be created to hold the string in each
 * message.
 *
 * Optionally the user may specify communication parameters for
 * tibrvTransport_Create.
 *
 * If none are specified the following defaults are used:
 *  service     "rendezvous" or "7500/udp"
 *  network     the result of gethostname
 *  daemon      "tcp:7500"
 *
 */

import java.util.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import com.tibco.tibrv.*;
import com.cisco.cns.nsm.client.*;

public class CiscoAppSend
{
    static int BUFLEN = 512;
    String subject = "cisco.mgmt.cns.config.load";
    String payload = "";
    String service = null;
    String network = null;
    String daemon  = null;
    String fileName = "config-file";

    String identifier = "D1";
    String appName = "config";
    int element = NSMClient.DEVICE;

    String FIELD_NAME = "DATA";

    boolean init=false;

    // Transport to rendezvous daemon
    TibrvTransport transport = null;
```

```
 /**
  *This method reads a payload string from a file, calls the namespace
  *mapper to map the original subject to a namespace specific subject,
  *and then publishes the payload on the mapped subject.
  *
  *  @param sub specifies the subject on which the application wishes to
  *     publish. This will be mapped using the NSM API calls
  *@param filename specifies the name of the file from which to read the
  *      XML payload
  *@param id specifies the identifier of the group/device
  *@param elem specifies whether the identifier refers to a group or a
  *      device
  */

 public int publish(String sub, String filename, String id, String app, int elem)
 {
 // Make sure the Tibco event mechanism is initialized
     if (!init)
 {
     System.out.println ("Not initialized. Call init method first. " );
     return -1;
 }
 // Set the Data members based on input parameters
     setDataMembers (sub, id,  app, elem, filename) ;

 // Read config from input file
 readConfigFromFile ();


 // Obtain mapping for subject from Namespace Mapper
 NSMResult nsmMapping = getMapping (appName, identifier, subject, NSMClient.PUBLISH,
element );
 if (nsmMapping != null)
 {

     // If mapping succeeded, publish on each of the mapped
     //  subjects

         NSMResultIterator iter = null;
     for (iter = new NSMResultIterator(nsmMapping); !iter.isEnd(); iter.advance())
     {
      String mappedSubject = iter.value();
           System.out.println("Mapped subject for publishing " + mappedSubject);
      publishMessage (mappedSubject, payload);
           }

 }
 return 0;
 }

/* This private method reads XML data from a file */
void readConfigFromFile ()
{
    if (fileName != null)
    {
         try
         {
             FileInputStream fileReader = new FileInputStream (fileName) ;

             byte [] buf = new byte [BUFLEN];


             while (fileReader.read (buf) != -1)
             {
```

```
System.out.println ("Buf is " + buf );

                String temp = new String (buf);

                     payload += temp;
                }
                System.out.println ("Configuration XML: " + payload);
       }
           catch (FileNotFoundException fne)
            {
                System.err.println("Failed to open " + fileName);
                fne.printStackTrace();
                System.exit(-1);
            }
            catch (IOException ie)
            {
                 System.out.println ("IOException occurred while reading from file " +
fileName);
                 ie.printStackTrace ();
                 System.exit(-1);
            }

       }
    }

    /* This private method sets the private data memebers of this class */
    void setDataMembers (String sub, String id, String app, int elem, String filename)
    {
        if (sub != null)
                subject = sub;
        if (id != null)
                identifier = id;
        if (app != null)
                appName = app;
        if (elem != 0)
                element = elem;
    if (filename != null)
        fileName = filename;

    }

    /* This private method actually publishes the mapped subject on the bus
    with the given payload */
    void publishMessage (String subject, String payload)
    {
        // Create the message
        TibrvMsg msg = new TibrvMsg();

        // Set send subject into the message
        try
        {
            msg.setSendSubject(subject);
        }
        catch (TibrvException e) {
            System.err.println("Failed to set send subject:");
            e.printStackTrace();
            System.exit(-1);
        }

        try
        {
          // Update the data portion of the message
              msg.update(FIELD_NAME,payload.getBytes(),TibrvMsg.OPAQUE);
```

```
      // Publish the message over the transport
          transport.send(msg);
    }
    catch (TibrvException e)
    {
        System.err.println("Error sending a message:");
        e.printStackTrace();
        System.exit(-1);
    }

}

/**
 * This method initializes the Tibco mechanism and creates a transport
 * between the application and the rendezvous daemon
 * @param service specifies the UDP port for rendezvous communication
 * @param network specifies the network interface for outbound messages
 * @param daemon specifies the TCP port on which the daemon listens for
 *       connections from applications
 */
void init (String sevice, String network, String daemon)
{

    // open Tibrv in native implementation
    try
    {
        Tibrv.open(Tibrv.IMPL_NATIVE);
    }
    catch (TibrvException e)
    {
        System.err.println("Failed to open Tibrv in native implementation:");
        e.printStackTrace();
        System.exit(-1);
    }

    try
    {
    // Create transport to rendezvous daemon
        transport = new TibrvRvdTransport(service,network,daemon);
    }
    catch (TibrvException e)
    {
        System.err.println("Failed to create TibrvRvdTransport:");
        e.printStackTrace();
        System.exit(-1);
    }

init = true;
}

/**
 * This method closes the transport and closes the Tibco event mechanism
 */
void fini ()
{

    // Close Tibrv, it will cleanup all underlying memory, destroy
    // transport and guarantee delivery.
    try
    {
        Tibrv.close();
    }
    catch(TibrvException e)
    {
```

```
            System.err.println("Exception dispatching default queue:");
            e.printStackTrace();
            System.exit(-1);
        }
    init=false;
    }

    /**
     * This private method contacts the Namespace Mapper and obtains a
     * namespace specific mapping for the subject
     * @param appName specifies the application namespace
     * @param id specifies the identifier of the device or group
     * @param subject specifies the subject to map
     * @param action specifies the operation : PUBLISH or SUBSCRIBE
     * @param element specifies the type that id refers to - GROUP or DEVICE
     */
    NSMResult getMapping (String appName, String id, String subject, int action, int
element)
    {
        int rc;
        NSMResult result = new NSMResult();

        //Creating a NSMClient instance
        NSMClient nsmc = NSMClient.create();

        //Attaching to an application called App1
        rc = nsmc.attach(appName);

        //Testing for the status of attach
        if(NSMClient.SUCCESS != rc)
        {
            System.out.println("we got an attach error");
            System.exit(-1);
        }

        //Calling the resolve method with parameters obtained above

        NSMResolveRequest request = new NSMResolveRequest(id, subject,
        action, element);

        rc = nsmc.resolve(result, request);

        //detaching the connection
        nsmc.detach();


        //Check to see the an empty vector gets returned.
        if(NSMClient.SUCCESS == rc)
        {
            System.out.println("Obtained mapping.");
        return result;
        }
        else
        {
            System.out.println("no result");
        return null;
        }

    }
}
```

Common Files

config_file — Sample Payload config.file

```
<?xml version="1.0" encoding="UTF-8" ?>
<config-event config-action="write" no-syntax-check="TRUE">
<identifier>12345</identifier>
<config-data>
<config-id>D1</config-id>
<cli>ip name-server 171.69.2.133</cli>
</config-data>
</config-event>
```

IMGW API Test Code

testDriver.Java

```
import com.cisco.mgmt.cns.IMGW.*;
import java.io.*;
import java.util.Vector;

public class testDriver
{
    public static void main(String[] args)
    {
        HopInfo aHopInfo = null;
        String  devType = null;
        String  devId = null;
        String  gatewayId = new String("G1");;
        String  ipAddress = null;
        int     port = 0;
        String  user = null;
        String  password = null;
        int i, j;

        try
        {
            //create a IMGWDevice object with specified URL
            IMGWDevice deviceOperator = new
IMGWDevice("http://penguin1/servlets/IMGWDeviceServlet");
            deviceOperator.setDebug(false);
            for (i=0; i<5; i++)
            {
                //create a device with given device-id and gateway-id
                devId = new String("176" + i);
                deviceOperator.createDevice(devId, gatewayId);

                //add one HopInfo to the device with given device-id
                devType = new String("IOS" + i + "k");
                ipAddress = new String("172.19.173.2" + i);
                port = 2001 + i;
                user = new String("lab" + i);
                password = new String("cisco" + i);
                aHopInfo = new HopInfo(devType, ipAddress, port, user, password);
                deviceOperator.addDeviceHop(devId, aHopInfo);

                //add second HopInfo to the device with given device-id
                devType = new String("CatIOS" + i + "k");
```

```
                ipAddress = null;
                port = 0;
                user = new String("pslab" + i);
                password = new String("xyz" + i);
                aHopInfo = new HopInfo(devType, ipAddress, port, user, password);
                deviceOperator.addDeviceHop(devId, aHopInfo);

            }

            deviceOperator.setDebug(false);
            for (i=0; i<5; i++)
            {
                aHopInfo = null;
                devId = new String("176" + i);
                System.out.println(devId + ":");

                //get HopInfo List
                for(j=0; j<2; j++)
                {
                    //get the jth HopInfo of the device with given device-id, j begins from
0
                    aHopInfo = deviceOperator.getDeviceHop(devId, j);
                    if(aHopInfo != null)
                    {
                        System.out.println(
                          aHopInfo.getDeviceType()+ "-->"
                        + aHopInfo.getIPAddr()+ "-->"
                        + aHopInfo.getPort()+ "-->"
                        + aHopInfo.getUserName()+ "-->"
                        + aHopInfo.getPassword());
                    }
                    else
                        System.out.println("None");
                }
            }
            deviceOperator.setDebug(false);
            // delete the device with given device-id
                         for (i=0; i<5; i++)
                         {
                                 devId = new String("176" + i);
                                 System.out.println(devId + ":");
                                 deviceOperator.deleteDevice(devId);
                         }

        }
        catch (InvalidHopException e)
        {
            System.err.println(e.toString());
        }
        catch (InvalidParameterException e)
                 {
                         System.err.println(e.toString());
                 }
        catch (NetworkException e)
                 {
                         System.err.println(e.toString());
                 }
        catch (NoSuchHopException e)
        {
            System.err.println(e.toString());
        }
        catch (DeviceExistsException e)
        {
            System.err.println(e.toString());
```

```
            }
            catch (DeviceNotFoundException e)
            {
                System.err.println(e.toString());
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

APPENDIX **B**

Sample Schema

For external directory, below are some input parameters such as LDAP container name and attributes that are needed when performing setup on the Cisco Configuration Engine.

```
Enter container name under which device objects are stored: [ou=CNSDevices]
Enter container name under which generic device objects are stored: [ou=GenericDevices]
Enter container name under which PIX device objects are stored: [ou=PIXDevices]
Enter container name under which ASA device objects are stored: [ou=ASADevices]
Enter container name under which linecard objects are stored: [ou=LinecardDevices]
Enter container name under which application objects are stored: [ou=CNSApplications]
Enter container name under which IMGW objects are stored: [ou=imgw]
Enter container name under which CIS objects are stored: [ou=CISObjects]
Enter container name under which image objects are stored: [ou=Images]
Enter container name under which CIS device objects are stored: [ou=CISDevices]
Enter container name under which distribution objects for Image are stored:
[ou=Distributions]
Enter container name under which Query objects are stored: [ou=Query]
Enter objectclass for device object: [IOSConfigClass]
Enter template attribute name in device objectclass: [IOSconfigtemplate]
Enter config ID attribute name in device objectclass: [IOSConfigID]
Enter event ID attribute name in device objectclass: [IOSEventID]
Enter device category attribute name in device objectclass: [AdminDevType]

Enabling Modular Router feature allows you to configure linecards
independently of the slot numbers.

Would you like to use Modular Router Feature (y/n)? [y] n
Enter Cisco-CE group attribute name in device objectclass: [parent]
```

Parameter Descriptions

Container name device object: The container in the directory under which device containers are created, e.g. ou=CNSDevices.

Container name generic device object: The container in the directory under which generic device objects are created.

Container name PIX device object: The container in the directory under which PIX device objects are created.

Container name ASA device object: The container in the directory under which ASA device objects are created.

Container name linecard device object: The container in the directory under which device objects are created.

Container name application object: The container in the directory under which application objects are created.

Container name IMGW object: The container in the directory under which IMGW objects are created.

Container name CIS object: The container in the directory under which CIS objects are created.

Container name image object: The container in the directory under which image objects are created.

Container name CIS device object: The container in the directory under which CIS device objects are created.

Container name distribution object: The container in the directory under which distribution objects are created.

Container name query object: The container in the directory under which query objects are created.

Device object class: The name of the user-defined object class for device object.

Device template attribute name: Attribute of the device class (as specified in the Object-class prompt) that specifies the template file for the device object. This is not the template file itself, just the name of the attribute that has the value of the template filename.

Device config ID attribute name: Attribute of the device class that uniquely identifies the device in the config-server domain.

Device event ID attribute name: Attribute of the device class that uniquely identifies the device in the event bus domain.

Device category attribute name: Attribute of the device class that identifies the device type.

Device group attribute name: The attribute of the device class (as specified in the Object-class prompt) that specifies the group(s) to which the device object belongs. This is only an attribute name, but not the groups themselves, e.g. parent.

APPENDIX C

IMGW Error Codes & Sample Source

IMGW retuns the following error messages when it cannot service requests because of failures or error conditions. The error string also contains an error number. The intent of error numbers is to help applications automate processing of error messages returned by IMGW.

For example in the error message: “E1050: Device Hop Info is Null” 1050 is the error number, while the string is for human readability only and is not part of the error string.

IMGW Error Code Messages

```
"E1040: Error"
"E1041: Failure"
"E1042: DeviceType Xml File not Found"
"E1043: Device Type Xml Parse Failure"
"E1044: Ace Data Block Null"
"E1045: Ace Message Block Null"
"E1046: Config Message Null"
"E1047: Config Message Params are Null or Missing"
"E1048: Config Message Temp File Creation Failure"
"E1049: Config Message Input File Creation Error"
"E1050: Device Hop Info is Null"
"E1051: Device Command Lookup Failure"
"E1052: Device Handler Spawn Failure"
"E1053: Config Log File Process Failure"
"E1054: Exec Message Null"
"E1055: Exec Message Params are Null or Missing"
"E1056: Exec Message Temp File Creation Failure"
"E1057: Exec Log File Process Failure"
"E1058: Device Id Message Null"
"E1059: Device Id Params are Null or Missing"
"E1060: Device Id Invalid Operation Specified"
"E1061: Transport Provider Creation Failure"
"E1062: Transport Session Creation Failure"
"E1063: ACE_Message_Queue getq returned NULL"
"E1064: Invalid subject"
"E1065: Operation failed"
"E1066: No appropriate callback"
"E1067: No appropriate config object for property"
"E1068: No DeviceId Message"
"E1068: No Device Id"
"E1069: Invalid Subject Identifier"
"E1070: No Config Response Message"
"E1071: No Exec Response Message"
"E1072: Exec Response Type not supported"
"E1073: Exec Invalid Response Type"
"E1074: Device Interface Thread Exiting"
```

"E1074: Device Interface Cannot Create Additional Thread(s)"
"E2040: Unable to connect to element - element did not respond or undefined prompt was displayed, check element ip address, port connection or username/password and customized user prompt."
"E2041: Unable to connect to communication server - communication server did not respond, check connected host ip address."
"E2042: Unable to connect to element - check login password."
"E2043: Unable to connect to communication server - check login password."
"E2044: Bad communication server port password."
"E2045: Unable to connect to element - check login username and password."
"E2046: Unable to connect to communication server - check login username and password."
"E2047: Unable to connect to element - communication server port is in use."
"E2048: Unable to connect to element - communication server port is not configured properly, TG does not support port username."
"E2049: Unable to enter element privileged command mode - check enable password."
"E2050: Could not enter element privileged command mode - check enable username and password."
"E2051: Configuration file is too big - check nvram size."
"E2052: Element operation did not complete within given time interval."
"E2053: Telnet process did not respond within given time interval - operation may not complete."
"E2055: Fail to clear communication server port."
"E2056: Cannot clear communication server port after maximum number of attempts, check and make sure that it is not in use by other users."
"E2057: Download successful, but found command line syntax errors."
"E2058: Download successful, but IOS parser may have found errors."
"E2059: Telnet process status lost due to socket communication error - check socket port usage by other applications in your system."
"E2060: IOS command failed."
"E2061: Local storage device: file-not-found error."
"E2062: The IOS command associated with this TG operation is not supported for this particular device and its IOS version."
"E2063: Local storage device: no-space-on-device error."
"E2064: Local storage device: device-not-available error."
"E2065: Local storage device: unspecified file I/O error."
"E2066: This Telnet Gateway service is not implemented."
"E2067: Dir local storage device: selected device or file is invalid."
"E2069: SSH Key is not Writable."
"E2070: Erase command error."
"E2071: Reload command error."
"E2072: Write memory command error."
"E2073: Show command error."
"E2074: IOS command execution error."
"E2075: Command authorization failed."
"E2076: Configuration memory has not been setup or has bad checksum."
"E2077: Connect via composite element failed."
"E2078: PIX Command Execution Error"
"E2079: PIX Authentication Failure"
"E2080: PIX Configuration Warning"
"E2081: PIX Configuration Error"
"E2082: Download successful, but found command line syntax errors."
"E2083: Command authorization failed."
"E2084: CATOS command execution error."
"E2085: Download successful, but CATOS parser may have found errors."
"E2086: Download successful, but found command line syntax errors."
"E2087: Download successful, but CSS parser may have found errors."
"E2088: Command authorization failed."
"E2089: CSS download failed."
"E2090: Download successful, but found command line syntax errors."
"E2091: Command authorization failed."
"E2092: CE command execution error."
"E2093: Download successful, but CE parser may have found errors."
"E2094: Download successful, but found command line syntax errors."
"E2095: Command authorization failed."

```
"E2096: CATIOS command execution error."
"E2097: Download successful, but CATIOS parser may have found errors."
"E2098: SSH connection error."
```

Code Sample for IMGW Device Information API

```
*-------------------------------------------------------------------
* APICmdTest.java - Test Device Info servlet via command line
 *
 * July 2001, Sanjay Aiyagari
 *
 * Copyright (c) 2001 by cisco Systems, Inc.
 * All rights reserved.
*-------------------------------------------------------------------
 * $Endlog$
 */
import com.cisco.cns.imgw.*;
import java.util.Vector;
import java.util.Enumeration;

class MyException extends Exception {
    public MyException() { super(); }
    public MyException(String s) { super(s); }
}

public class APICmdTest
{
    public static void usage ()
    {
        System.out.print(
"Usage: APICmdTest <servlet_url> create <devid> <gateway_id> [debug]\n" +
"       APICmdTest <servlet_url> add <devid> <dev_type> <ip> <port> <user>
<passwd> [debug]\n" +
"       APICmdTest <servlet_url> get <devid> <num> [debug]\n" +
"       APICmdTest <servlet_url> delete <devid> [debug]\n" +
"       APICmdTest <servlet_url> list [<gateway_id>] [debug]\n"
            );
    }

    public static void main (String args[])
    {
        try
        {
            int arg_length = args.length;
            if (arg_length < 2) {
                throw new MyException("Not enough arguments specified");
            }
            IMGWDevice device = new IMGWDevice(args[0]);
            if (args[arg_length - 1].equalsIgnoreCase("debug")) {
                device.setDebug(true);
                arg_length--;
            }
            if (args[1].equalsIgnoreCase("create")) {
                if (arg_length < 4) {
                    throw new MyException(
                    "CREATE requires <devid> and <gateway_id> argument");
                }
                device.createDevice(args[2], args[3]);
            } else if (args[1].equalsIgnoreCase("add")) {
                if (arg_length < 8) {
                    throw new MyException("ADD requires more arguments");
                }
```

```
                HopInfo hopInfo = new HopInfo(
                    args[3], args[4], Integer.parseInt(args[5]), args[6],
args[7]);
                    //Preprocess.parameter(args[3]),
Preprocess.parameter(args[4]), Integer.parseInt(args[5]),
                    //Preprocess.parameter(args[6]),
Preprocess.parameter(args[7]));
                device.addDeviceHop(args[2], hopInfo);
            } else if (args[1].equalsIgnoreCase("get")) {
                if (arg_length < 4) {
                    throw new MyException("GET requires more arguments");
                }
                HopInfo record =
                    device.getDeviceHop(args[2], Integer.parseInt(args[3]));
                System.out.println(
                     "devtype=" + record.getDeviceType() +
                    ",ipaddr=" + record.getIPAddr() +
                    ",port=" + Integer.toString(record.getPort()) +
                    ",username=" + record.getUserName() +
                    ",password=" + record.getPassword());
            } else if (args[1].equalsIgnoreCase("delete")) {
                if (arg_length < 3) {
                    throw new MyException("DELETE requires <devid>
argument");
                }
                device.deleteDevice(args[2]);
            } else if (args[1].equalsIgnoreCase("list")) {
                Vector devlist = null;
                if (arg_length == 2) {
                    devlist = device.listAllDevices();
                } else {
                    devlist = device.listAllDevices(args[2]);
                }
                Enumeration devenum = devlist.elements();
                while (devenum.hasMoreElements()) {
                    System.out.println(devenum.nextElement().toString());
                }
            } else {
                throw new MyException("Invalid argument");
            }
        }
        catch (Exception e)
        {
            if (e instanceof MyException) {
                System.out.println(e.getMessage());
                usage();
                System.exit(1);
            } else if (e instanceof NetworkException) {
                System.out.println(e.getMessage());
                System.exit(2);
            } else if (e instanceof NoSuchHopException) {
                System.out.println(e.getMessage());
                System.exit(3);
            } else if (e instanceof InvalidHopException) {
                System.out.println(e.getMessage());
                System.exit(4);
            } else if (e instanceof InvalidParameterException) {
                System.out.println(e.getMessage());
                System.exit(5);
            } else if (e instanceof DeviceNotFoundException) {
                System.out.println(e.getMessage());
                System.exit(6);
            } else if (e instanceof DeviceAlreadyExistsException) {
                System.out.println(e.getMessage());
```

```
                System.exit(7);
            } else {
                e.printStackTrace();
            }
        }
    }
}
```

INDEX

Symbols

A

C

D

E

G

H

I

J

L

M

N

O

P

R

S

T

X