Cisco IP Solution Center API Programmer Guide, 4.1

Corporate 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: 408 526-4100

Customer Order Number:
Text Part Number: OL-7649-01

Cisco IP Solution Center API Programmer Guide, 4.1

INDEX

About This Guide

The *Cisco IP Solution Center API Programmer Guide, 4.1* describes APIs that are available to third party Operations Support System (OSS) products. The ISC API provides a mechanism for inserting, retrieving, updating, and removing data from the ISC servers using an eXtensible Markup Language (XML) interface.

This preface contains the following sections:

Who Should Use This Book

This guide is intended to be a technical resource for application developers who want to use the ISC API to provision network services.

You should have an advanced level of understanding of Internet network design, operation, and terminology, be familiar with Cisco Internetwork Operating System (IOS) software and its commands, and have a basic understanding of the Cisco IP Solution Center (ISC) product.

Service provider developers that use the API should also have an understanding of a high-level programming language such as Java, or an equivalent language. Additionally, we recommend that you have knowledge of the following:

- Hypertext Transport Protocol (HTTP/HTTPS)
- Socket programming
- XML and XML Schema

You should be familiar with the ISC GUI and how to use it to provision your network. In most cases, API operations correlate to GUI operations. See Appendix A, "GUI to API Mapping," for more information.

How This Book Is Organized

This programmer guide is organized as follows:

- These chapters describe the ISC API fundamentals.
  - Chapter 1, "Introduction to the ISC API," describes the fundamental concepts for using the XML API, and includes the API model, components, and operations.
  - Chapter 2, "Getting Started," describes the steps required to log into the API, and the basic steps for creating inventory, service requests, and service policies.
  - Chapter 3, "Common APIs," describes the APIs for operations that are common to all ISC services.
  - Chapter 4, "Using Templates," describes ISC templates and how to use them to download configurations and use them in a service request.
  - Chapter 5, "Monitoring APIs," describes how to use the API to monitor services and create reports.
- These chapters describe the concepts for each service and provide a provisioning example, which includes the steps required to provision the services using the ISC API.
  - Chapter 6, "MPLS Provisioning."
  - Chapter 7, "L2VPN Provisioning."
  - Chapter 8, "VPLS Provisioning."
  - Chapter 9, "QoS Provisioning."
- Additional information:
  - Appendix A, "GUI to API Mapping," maps the GUI operations to the corresponding APIs.
  - Appendix B, "Implementing a Notification Server," provides an example for implementing a servlet to receive notification events from ISC, and lists all events that can be collected for notification.
  - Appendix C, "Scripts," provides the list of scripts available with the ISC software application.

Related Documentation

The entire documentation set for Cisco IP Solution Center, 4.1 can be accessed at:

http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1

The following documents comprise the ISC 4.1 documentation set.

General documentation (in suggested reading order):

- *Cisco IP Solution Center Getting Started and Documentation Guide, 4.1*

  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/docguide/index.htm
- *Release Notes for Cisco IP Solution Center, 4.1*

http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/relnotes/index.htm

- *Cisco IP Solution Center Installation Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/install/index.htm
- *Cisco IP Solution Center Infrastructure Reference, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/infrastr/index.htm
- *Cisco IP Solution Center System Error Messages, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/mess/index.htm

Application and technology documentation (listed alphabetically):

- *Cisco IP Solution Center L2VPN User Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/l2vpn/index.htm
- *Cisco IP Solution Center MPLS VPN User Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/mpls/index.htm
- *Cisco IP Solution Center Quality of Service User Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/qos/index.htm
- *Cisco IP Solution Center Traffic Engineering Management User Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/tem/index.htm
- *Cisco MPLS Diagnostics Expert 1.0 User Guide on ISC 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/trble/index.htm

API Documentation:

- *Cisco IP Solution Center API Programmer Guide, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_gd/index.htm
- Index: *Cisco IP Solution Center API Programmer Reference, 4.1*
  http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/index.htm

Note All documentation *might* be upgraded over time. All upgraded documentation will be available at the same URLs specified in this document.

Developer Services

The Cisco Developer Support Program provides a central resource point for API users to get additional services such as training, FAQs, support, and consulting. For more information, see the following URL:

http://www.cisco.com/en/US/products/svcs/ps3034/ps5408/ps5418/serv_home.html

Developer Services for ISC include a download page from which independent software vendors (ISVs), partners, and customers can download sample repositories, API code examples, and get links to documentation. For more information, see the following URL:

http://www.cisco.com/cgi-bin/front.x/cse/DSFC/SubsecTopic.pl?secID=74&secName=Network%20Management%20-%20OSS%2FBSS

Additional Information

- For Tomcat web server:

  http://jakarta.apache.org/tomcat/index.html
- For SOAP plug-in:
  - http://xml.apache.org/soap/
- For XML and XML Schema
  - http://www.w3.org/XML
  - http://www.w3.org/TR/xmlschema-0/
- For events and notifications through the CIM event model

  http://www.dmtf.org/standards/documents/CIM/DSP0107.pdf
- For CIM Objects over HTTP, DMTF

  http://www.dmtf.org/standards/documents/WBEM/DSP200.html

Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation at this URL:

http://www.cisco.com/techsupport

You can access the Cisco website at this URL:

http://www.cisco.com

You can access international Cisco websites at this URL:

http://www.cisco.com/public/countries_languages.shtml

Product Documentation DVD

Cisco documentation and additional literature are available in the Product Documentation DVD package, which may have shipped with your product. The Product Documentation DVD is updated regularly and may be more current than printed documentation.

The Product Documentation DVD is a comprehensive library of technical product documentation on portable media. The DVD enables you to access multiple versions of hardware and software installation, configuration, and command guides for Cisco products and to view technical documentation in HTML. With the DVD, you have access to the same documentation that is found on the Cisco website without being connected to the Internet. Certain products also have .pdf versions of the documentation available.

The Product Documentation DVD is available as a single unit or as a subscription. Registered Cisco.com users (Cisco direct customers) can order a Product Documentation DVD (product number DOC-DOCDVD=) from Cisco Marketplace at this URL:

http://www.cisco.com/go/marketplace/

Ordering Documentation

Beginning June 30, 2005, registered Cisco.com users may order Cisco documentation at the Product Documentation Store in the Cisco Marketplace at this URL:

http://www.cisco.com/go/marketplace/

Nonregistered Cisco.com users can order technical documentation from 8:00 a.m. to 5:00 p.m. (0800 to 1700) PDT by calling 1 866 463-3487 in the United States and Canada, or elsewhere by calling 011 408 519-5055. You can also order documentation by e-mail at tech-doc-store-mkpl@external.cisco.com or by fax at 1 408 519-5001 in the United States and Canada, or elsewhere at 011 408 519-5001.

Documentation Feedback

You can rate and provide feedback about Cisco technical documents by completing the online feedback form that appears with the technical documents on Cisco.com.

You can send comments about Cisco documentation to bug-doc@cisco.com.

You can submit comments by using the response card (if present) behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Customer Document Ordering
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Cisco Product Security Overview

Cisco provides a free online Security Vulnerability Policy portal at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

From this site, you can perform these tasks:

- Report security vulnerabilities in Cisco products.
- Obtain assistance with security incidents that involve Cisco products.
- Register to receive security information from Cisco.

A current list of security advisories and notices for Cisco products is available at this URL:

http://www.cisco.com/go/psirt

If you prefer to see advisories and notices as they are updated in real time, you can access a Product Security Incident Response Team Really Simple Syndication (PSIRT RSS) feed from this URL:

http://www.cisco.com/en/US/products/products_psirt_rss_feed.html

Reporting Security Problems in Cisco Products

Cisco is committed to delivering secure products. We test our products internally before we release them, and we strive to correct all vulnerabilities quickly. If you think that you might have identified a vulnerability in a Cisco product, contact PSIRT:

- Emergencies—security-alert@cisco.com

  An emergency is either a condition in which a system is under active attack or a condition for which a severe and urgent security vulnerability should be reported. All other conditions are considered nonemergencies.

- Nonemergencies—psirt@cisco.com

In an emergency, you can also reach PSIRT by telephone:

- 1 877 228-7302
- 1 408 525-6532

Tip We encourage you to use Pretty Good Privacy (PGP) or a compatible product to encrypt any sensitive information that you send to Cisco. PSIRT can work from encrypted information that is compatible with PGP versions 2.*x* through 8.*x*.

Never use a revoked or an expired encryption key. The correct public key to use in your correspondence with PSIRT is the one linked in the Contact Summary section of the Security Vulnerability Policy page at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

The link on this page has the current PGP key ID in use.

Obtaining Technical Assistance

Cisco Technical Support provides 24-hour-a-day award-winning technical assistance. The Cisco Technical Support & Documentation website on Cisco.com features extensive online support resources. In addition, if you have a valid Cisco service contract, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not have a valid Cisco service contract, contact your reseller.

Cisco Technical Support & Documentation Website

The Cisco Technical Support & Documentation website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day, at this URL:

http://www.cisco.com/techsupport

Access to all tools on the Cisco Technical Support & Documentation website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

http://tools.cisco.com/RPF/register/register.do

Note Use the Cisco Product Identification (CPI) tool to locate your product serial number before submitting a web or phone request for service. You can access the CPI tool from the Cisco Technical Support & Documentation website by clicking the **Tools & Resources** link under Documentation & Tools. Choose **Cisco Product Identification Tool** from the Alphabetical Index drop-down list, or click the **Cisco Product Identification Tool** link under Alerts & RMAs. The CPI tool offers three search options: by product ID or model name; by tree view; or for certain products, by copying and pasting **show** command output. Search results show an illustration of your product with the serial number label location highlighted. Locate the serial number label on your product and record the information before placing a service call.

Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool provides recommended solutions. If your issue is not resolved using the recommended resources, your service request is assigned to a Cisco engineer. The TAC Service Request Tool is located at this URL:

http://www.cisco.com/techsupport/servicerequest

For S1 or S2 service requests or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)
EMEA: +32 2 704 55 55
USA: 1 800 553-2447

For a complete list of Cisco TAC contacts, go to this URL:

http://www.cisco.com/techsupport/contacts

Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

Severity 1 (S1)—Your network is “down,” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Severity 2 (S2)—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Severity 3 (S3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- Cisco Marketplace provides a variety of Cisco books, reference guides, documentation, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

  http://www.cisco.com/go/marketplace/

- *Cisco Press* publishes a wide range of general networking, training and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:

  http://www.ciscopress.com

- *Packet* magazine is the Cisco Systems technical user magazine for maximizing Internet and networking investments. Each quarter, Packet delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can access Packet magazine at this URL:

  http://www.cisco.com/packet

- *iQ Magazine* is the quarterly publication from Cisco Systems designed to help growing companies learn how they can use technology to increase revenue, streamline their business, and expand services. The publication identifies the challenges facing these companies and the technologies to help solve them, using real-world case studies and business strategies to help readers make sound technology investment decisions. You can access iQ Magazine at this URL:

  http://www.cisco.com/go/iqmagazine

  or view the digital edition at this URL:

  http://ciscoiq.texterity.com/ciscoiq/sample/

- *Internet Protocol Journal* is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

  http://www.cisco.com/ipj

- Networking products offered by Cisco Systems, as well as customer support services, can be obtained at this URL:

  http://www.cisco.com/en/US/products/index.html

- Networking Professionals Connection is an interactive website for networking professionals to share questions, suggestions, and information about networking products and technologies with Cisco experts and other networking professionals. Join a discussion at this URL:

  http://www.cisco.com/discuss/networking

- World-class networking training is available from Cisco. You can view current offerings at this URL:

  http://www.cisco.com/en/US/learning/index.html

Introduction to the ISC API

The Cisco IP Solution Center (ISC) application program interface (API) allows you to use operations support system (OSS) client programs to connect to the ISC system. The ISC APIs provide a mechanism for inserting, retrieving, updating, and removing data from ISC servers using an eXtensible Markup Language (XML) interface request/response system. The ISC API optionally uses Secure Hypertext Transfer Protocol (HTTPS) for message encryption, and Cisco role-based access control (RBAC) for user authentication.

The ISC APIs use an HTTP/HTTPS/SOAP (Simple Object Access Protocol) interface. The API requests are executed using a combination of HTTP/HTTPS and SOAP by sending the XML data to the API server. The server returns an XML response, which is also an encoded SOAP message, to indicate if the request is successful, or to return data.

The API optionally uses a notification server for database change events. An event is registered and a notification is sent any time a database object is created, modified, or deleted, or when a scheduled task begins or ends its execution. Event notifications are sent in the form of an XML response to the client, or to a specified URL.

You can use the API to perform the same operations as you would using the ISC GUI. For more information, see Appendix A, “GUI to API Mapping.”

This guide describes how to use the API to perform operations that are common to all ISC services and provides examples for provisioning MPLS, L2VPN, VPLS, and QoS services in your network.

This chapter contains the following sections:

API Components

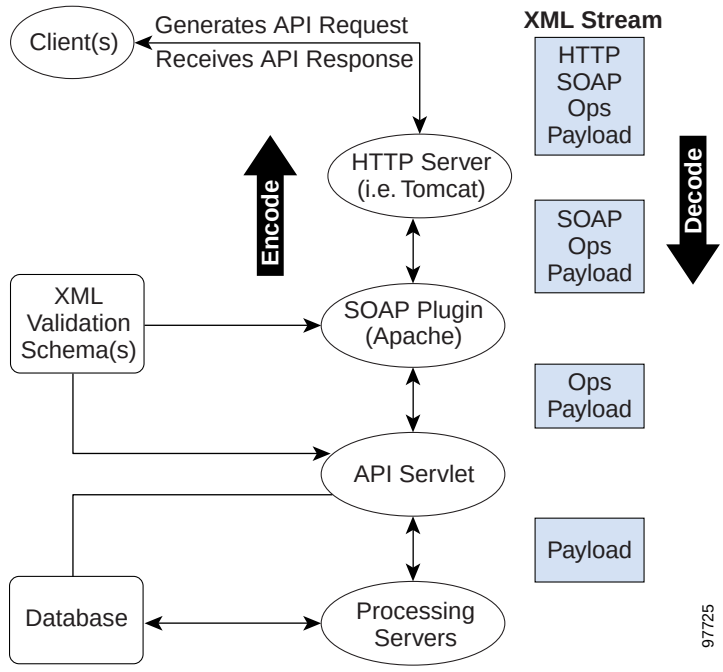
The main components of the ISC API are:

- Client—The OSS client program.
- HTTP/HTTPS Server—A standard HTTP/HTTPS Tomcat server to process the HTTP/HTTPS binding information.
- SOAP Plug-In—A standard Apache SOAP plug-in.

- API Server/Servlet—Receives SOAP messages and removes the SOAP encoding.
- API Notification Server—Used for asynchronous notifications for database change events. ISC learns of events that occur using the Tibco Event Bus.
- Processing Servers—The servers that perform the specific processing activities.
- Database—The ISC repository.
- XML validation schema(s) and metadata—Validation files for the XML encoded data.

Figure 1-1 shows the main components of the ISC API and the process flow for XML messages.

Figure 1-1 API Components



These components are described in the following sections.

Client

The client can be any OSS client program. It formulates the XML request messages and receives the XML responses. Use any language that supports the XML format to generate the API messages.

The client interface:

- Logs into the ISC API system
- Generates XML requests
- Sends requests to the API server
- Receives responses from the API server
- Parses the XML response data content

Note The API client should handle unrecoverable exceptions, that is ConnectionException, by itself to ensure a successful sequential execution.

HTTP/HTTPS Server

ISC uses a Tomcat server to process the HTTP/HTTPS binding information. The default ports are:

- HTTP—8030
- HTTPS—8443

You can specify a different port during the ISC installation. Refer to the *Cisco IP Solution Center Installation Guide* for more information.

HTTP Transport

The API uses standard HTTP/HTTPS for message transport. The payload of an HTTP request or response is a SOAP message. Each SOAP request is sent to the web server using HTTP POST. The following are required HTTP headers:

- POST —The first header identifies that this particular POST is intended for the SOAP API. All HTTP requests that do not include a POST are ignored.
- Content-type: text/xml—The second header confirms that the data being sent is XML. If this header is not found, an HTTP 415 error is returned.
- Content-length: <value in kilobytes>—The third header must be a positive integer and cannot exceed 40. If the value is greater than 40 kilobytes, an HTTP 413 error is returned.
- The fourth header is the length (in bytes) of the SOAP message.

The following is an example HTTP header for an XML request:

```
POST /soap/servlet/messagerouter HTTP/1.0
Host: server1.myhost.com:80
Content-type: text/xml
Content-length: 613
```

Note HTTP headers might vary. Refer to the client software included with the ISC installation for the latest HTTP software that shows the HTTPS strings.

HTTP Response

If an error is detected in the HTTP protocol, the appropriate HTTP error message is returned in the HTTP response. The following are examples of HTTP return codes that can be returned for processing a SOAP request:

- Content length exceeds 40 KB

  ```
  HTTP/1.1 413 Request Entity Too Large
  ```

- Content type is not "text/xml"

  ```
  HTTP/1.1 415 Unsupported Media Type
  ```

- Request method is any method other than POST

```
HTTP/1.1 405 Method Not Allowed
```

During the processing of a SOAP request, you always receive the HTTP return code “HTTP/1.1 200 OK”, whether an error occurs or not. See the following example:

```
contacting: http://myserver.com:8030/soap/servlet/messagerouter
with:       /tmp/tmp.2764
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 597
Date: Fri, 21 Feb 2003 15:43:58 GMT
Server: Apache Tomcat/4.0.1 (HTTP/1.1 Connector)
Set-Cookie: JSESSIONID=C2337537E4C568A7A6228022A3A39521;Path=/soap
```

HTTP Authentication/Encryption

HTTP authentication is optional and is controlled through the HTTP basic authentication scheme. You must deactivate anonymous access to enforce user authentication.

HTTPS (HTTP Secure Socket Layer (SSL)) can be used to encrypt the API message. SSL is not enabled on the server by default. To use SSL, you must install HTTPS during the ISC installation process. When the SSL certificate is installed on the server, you can send requests using the HTTPS protocol instead of HTTP.

Note The ISC API supports remote authentication. See the “Remote Authentication” section on page 5-2 for more information.

SOAP Plug-in

ISC includes a SOAP plug-in for validating that messages comply with the SOAP protocol. SOAP is an XML-based protocol that consists of:

- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.
- A framework for describing what is in a message and how to process it.

SOAP provides server-side infrastructure for deploying, managing and running SOAP enabled services.

Note The ISC API supports SOAP formatting through SOAP libraries. However, SOAP libraries can be disabled if necessary for service provider clients that do not require SOAP library capabilities. Without this functionality, ISC uses normal HTTP/HTTPS socket mechanisms to send and receive SOAP formatted messages. To disable SOAP libraries, set **nbi.Writer.SoapEncapsulation=false** (the default setting) in the ISC properties file. The default setting allows you to run both SOAP-encapsulated and non-SOAP-encapsulated clients. You can set this attribute to **true** if you are running a pure SOAP environment.

SOAP Messages

The payload of an HTTP request/response is a SOAP message. A SOAP message includes the envelope, the header, and the body.

- The **soapenv-Envelope** defines a framework for describing what is in a SOAP message and how to process it.
- The **soapenv-Header** defines session data and contains message handling information and information about the format of the payload data.
- The **soapenv-Body** element contains the child elements (operations, name/value pairs, key properties), which are the domain specific data.
  - The **soapenv-Fault** element contains error messages that are particular to SOAP. These messages are returned in the SOAP body.

SOAP Message Envelope

The message envelope is used to declare namespaces. SOAP messages are routed using the XML namespaces associated with the first element in the message body. The first block of namespaces in the SOAP Envelope are standard for SOAP and XML encoding. See the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns0="http://www.cisco.com/cim-cx/2.0"
  xmlns:ns1="http://insmbu.cisco.com/urn:CIM">
  <soapenv:Header>
   <ns0:message id="87855" timestamp="2002-12-13T14:55:38.885Z"
                sessiontoken="p36bttjwy1"/>
  </soapenv:Header>
  <soapenv:Body>
   <ns1:createInstanceResponse/>
  </soapenv:Body>
</soapenv:Envelope>
Responses
```

For the namespaces indicated in bold:

- xmlns:ns0=”http://www.cisco.com/cim-cx/2.0” is used to indicate the message header formatting.
- xmlns:ns1="http://insmbu.cisco.com/urn:CIM" is used to namespace the operations performed and the data model.

SOAP Message Header

The message header includes information about the message itself. This includes the message ID, the timestamp for the message, the session token, and wait flags. The following example shows SOAP header information:

```
<soapenv:Header>
   <ns0:message id="87855" timestamp="2002-12-13T14:55:38.885Z"
                sessiontoken="p36bttjwy1" wait="true" waitTimeout="60" />
  </soapenv:Header>
  <soapenv:Body>
```

Table 1-1 describes the details of a SOAP message header.

Table 1-1 Header Definition

Element	Description
Message ID	A correlation ID used for tracking client requests and responses. ISC ignores this ID.
Timestamp	Time when message was sent (in Zulu).
Session Token	Session ID assigned during the login and used to access the system.
Wait Flags	Described in Table 1-2.

Wait flags are specified in the SOAP message header and in certain view operations. Table 1-2 lists the wait flags that can be returned in an XML response. Wait flags are optional request attributes.

Table 1-2 SOAP Message Wait Flags

Flag	Applies to	Values	Comments
level	enumerateInstance	positive integer	The object depth that is returned in a view. Suppresses lower level objects if needed.
wait	Header	true | false	Specifies whether the connection should stay open until the service request completes. Upon completion, the state of the service request is returned. **Default=false** (no wait).
waitTimeout	Header	Interval, in seconds	Maximum time to wait for a service request to complete. You can set the **waitTimeout** value in the ISC properties file. The default is 20 minutes. **Note** If the wait times out, the service request returns an error message indicating that the wait time has been exceeded. However, the request is still processed.

Tip To use the API to lock a device so that ISC cannot access it for provisioning, see the "Device Locking" section on page 5-21.

SOAP Message Body

The message body within a SOAP envelope implements a set of operations. The first line of the SOAP body is the method call, or operation, and the object for this operation is indicated by the **className**. Attributes for an object are specified in the properties (name/value pairs) for each class.

In the following XML example for creating a new site, the operation is **createInstance** and the **className** is **Site**. Properties for **Name**, **Organization**, and **SiteInfo** are included.

```
<soapenv:Body>
   <ns1:createInstance>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">Site</className>
     <properties xsi:type="ns1:CIMPropertyList"
                 soapenc:arrayType="ns1:CIMProperty[]">
```

```
    <item xsi:type="ns1:CIMProperty">
     <name xsi:type="xsd:string">Name</name>
      <value xsi:type="xsd:string">Site1</value>
     </item>
    <item xsi:type="ns1:CIMProperty">
     <name xsi:type="xsd:string">Organization</name>
      <value xsi:type="xsd:string">Customer2</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
     <name xsi:type="xsd:string">SiteInfo</name>
      <value xsi:type="xsd:string">Site comment info</value>
     </item>
    </properties>
   </objectPath>
  </ns1:createInstance>
 </soapenv:Body>
```

See the "Operations" section on page 1-9 for more information on operations implemented in the SOAP body.

Message Validation/SOAP Faults

If an XML request is not well-formed, or if there is an internal error in the SOAP server, you receive a SOAP *Fault* message. See the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns0="http://www.cisco.com/cim-cx/2.0"
  xmlns:ns1="http://insmbu.cisco.com/urn:CIM">
  <soapenv:Header>
   <ns0:message id="87855" timestamp="2002-12-13T14:55:38.885Z"
                sessiontoken="p36bttjwy1"/>
  </soapenv:Header>
  <soapenv:Body>
           < soapenv:Fault>
            <faultcode>SOAP-ENV:Client</faultcode>
            <faultstring>Client Error</faultstring>
            <faultactor/>
        </soapenv:Fault>
  </soapenv:Body>
```

Additionally, if the XML request fails the validation, an error is generated.

- Exception messages are shown to the user within the <Exception> XML tags in the XML response.
- Frequently seen error messages are translated to reader-understandable text and presented within the <Message> tag of the XML response.

For ISC error reporting, see the "API Error Messages" section on page 1-16.

Message Security

The ISC API supports the following security methods for SOAP messages:

- For message security, the API supports encryption at the transport layer. See the "HTTP Response" section on page 1-3.

- For user security, the API supports Cisco role-based access control (RBAC) to control user sessions. See the "Tasks" section on page 3-8 for more information.

API Notifications Server

This server is used for asynchronous notifications for database change events. It listens for the specified database change events and sends a notification across the client connection or to a URL.

Note The notification URL is set in the ISC properties file and can be specified during installation.

See the "Event Notifications" section on page 5-1 for more information.

API Server/Servlet

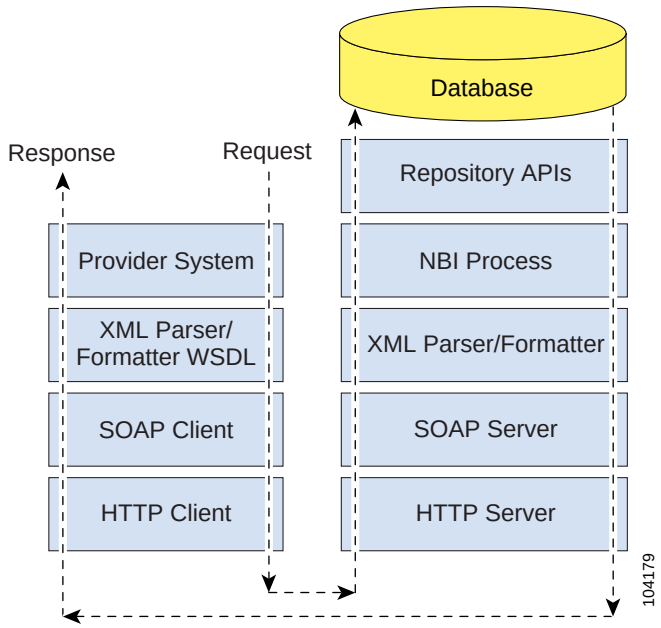
The API server (running as a servlet) receives the SOAP messages and removes the SOAP encoding. The API server also validates that the message is formatted correctly before initiating processing.

- For requests, the API delegates the request message to the appropriate processing server.
- For responses, the processing server sends the request back to the client.

The API is synchronous for operations, meaning a request is issued and then a response for each request is issued. An HTTP/HTTPS connection is established for incoming requests and another HTTP/HTTPS connection is used for receiving outgoing responses. You can disconnect and re-establish these connections as needed.

Figure 1-2 shows the process flow for an XML request from the client program in the provider system to the ISC repository, and the return path for the XML responses.

Figure 1-2 Process Flow Diagram



Operations

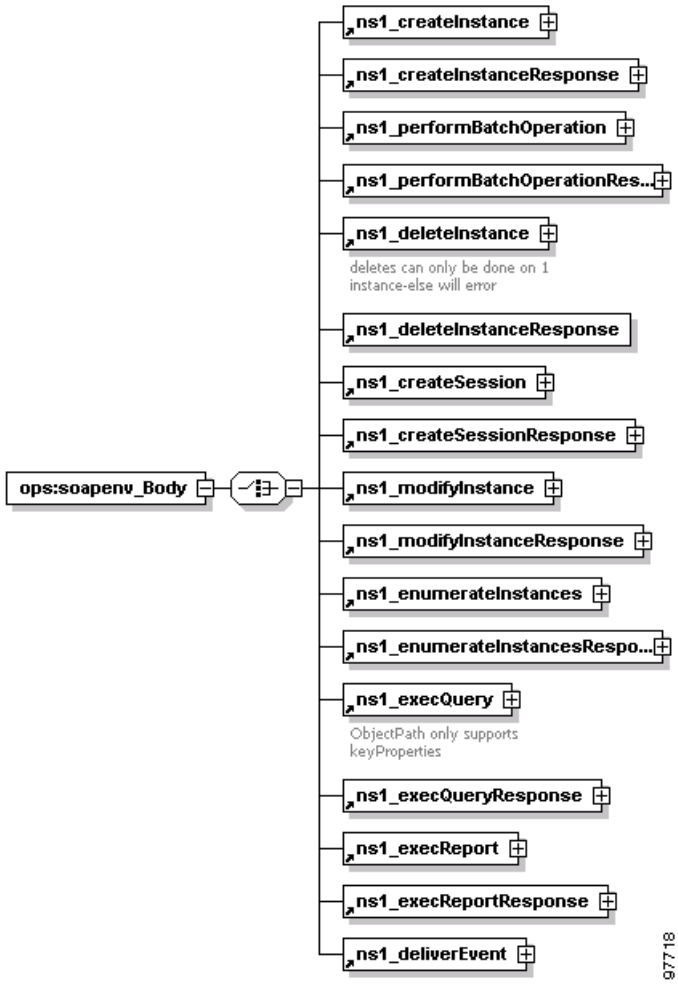
The process servers perform the specific processing activities, or operations. These operations are executed on ISC inventory and service objects. The API repository object model contains all object relationships, attributes, and operations.

API operations are divided into three categories: general, specialized, and response.

- General operations are executed on ISC inventory objects.
  - createInstance—Create an object
  - deleteInstance—Remove an object
  - modifyInstance—Edit an object
  - execQuery—SQL-based queries
  - execReport—Canned reports and SLA report queries
  - execMethod—Used for device locking.
  - enumerateInstances—Get multiple objects, or view the properties of an object
- Specialized operations are used for accessing the system.
  - createSession—Login
  - deleteSession—Logout
- Each API operation has an associated response (except **deliverEvent,** which is a response itself). There are four types of responses for each of the operations:
  - Response—Response to indicate that a request was successful.
  - Data—Response for a request for information.
  - Notifications (**deliverEvent**)—Response to indicate the addition, deletion, or change to a database object.
  - Errors—Response to indicate that an error has occurred.

The operations that can be executed for ISC repository objects are listed in Figure 1-3.

Figure 1-3 SOAP Envelope Body Operations



See the appropriate chapter in this guide for more information on APIs for specific operations.

XML Schema

ISC uses an XML schema and metadata to validate that the XML requests passed from the client are correct. The validation verifies that the **className** is valid and that the attributes listed in the XML request are recognized.

The API XML schema is defined by the World Wide Web Consortium (W3C) organization, which defines a structured way to express data structures. The schema provides constructs for defining data types and the mapping of those data types to data structures, including namespace definitions. ISC uses namespaces to separate the blade services and the SOAP data structures.

Note The inventory of XML examples for the ISC API is located at: http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

The API namespaces are for W3C, SOAP, CIM operations, and ISC services. The ISC services are separated into blades in the schema. The following are namespace imports for the ISC service order schema:

- <xsd:import namespace="http://isc.cisco.com/qos" schemaLocation="qos.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/l2vpn" schemaLocation="l2vpn.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/mpls" schemaLocation="mpls.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/template" schemaLocation="template.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/common" schemaLocation="common.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/task" schemaLocation="task.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/sla" schemaLocation="sla.xsd"/>
- <xsd:import namespace="http://isc.cisco.com/vpls" schemaLocation="vpls.xsd"/>

The following example is a partial schema for an L2VPN service definition. This example shows the element name and attributes for **ServiceDefinition**, **VPN**, and **EndToEndWire**. **ServiceDefinition** and **VPN** are **type=string**.

Mandatory elements are indicated by the property **minOccurs=1**. If **minOccurs=0**, the element is optional. If neither is specified, the default setting is **minOccurs=1** and **maxOccurs=1**.

```
<xsd:element name="ServiceRequestDetails">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ServiceDefinition" type="xsd:string" />
          <xsd:element name="VPN" type="xsd:string" />
          <xsd:element name="EndToEndWire" type="isc:EndToEndWire" minOccurs="1"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

Enumerated Schemas

Certain XML schema elements require that you specify enumerations. The enumeration types are specified in the schema. In the following example for the **className PE** (PE device), there are four enumeration values for the subelement **Role**.

```
<xsd:element name="Role" minOccurs="0">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="PE_POP"/>
      <xsd:enumeration value="PE_CLE"/>
      <xsd:enumeration value="PE_CORE"/>
      <xsd:enumeration value="PE_MVRF"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

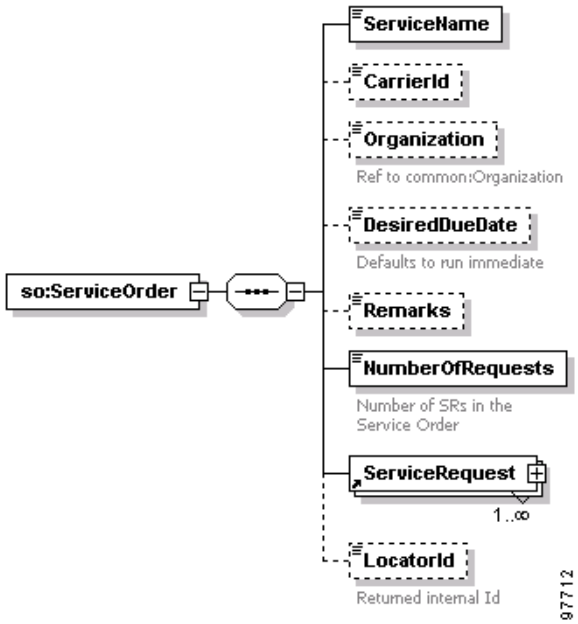
If the element **Role** is set to mandatory (**minOccurs=1**), you must specify one enumeration value for **Role** in the XML request for **className PE**.

Viewing Schema

The XML schemas can be viewed using a third-party schema browser such as XMLSPY©. In schema diagrams, objects shown with a solid line are mandatory, and objects shown with a dotted line are optional. Annotations in the XML schema are listed below the object, if needed. The plus (+) sign on an object indicates it has a child object. To view the child objects, click the plus sign using XMLSPY© or while viewing in another schema browser.

Figure 1-4 shows a schema diagram for a service order.

Figure 1-4 Schema Diagram for Service Order



In this schema diagram for a service order, note that the **ServiceName**, **NumberOfRequests**, and **ServiceRequest** objects are required, and that **ServiceRequest** has child objects.

ISC provides the following schemas with the product:

- common.xsd
- l2vpn.xsd
- mpls.xsd
- operations.xsd
- qos.xsd
- reports.xsd
- root.xsd (entire schema)
- serviceorder.xsd
- sla.xsd
- task.xsd
- template.xsd

- vpls.xsd

XML Examples

ISC provides example XML requests and responses with the product. Use the XML examples as a reference to develop your own client code.

The inventory of XML examples for the ISC API can be found at the following location:

http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

- When you view the XML examples using Internet Explorer, you see them in collapsible view. Collapsible view allows you to hide lower level objects and attributes by clicking on the (-) sign next to the CIM object path or property.
- When you view the XML examples using Netscape, you see them in text-only view.

Table 1-3 describes the different categories for XML examples and where each is described in this guide.

Table 1-3 XML Examples Available with ISC

Example XML Category	Described in
Events	Chapter 3, “Common APIs.”
ExecQuery	Chapter 3, “Common APIs.”
General	Chapter 3, “Common APIs.”
Inventory	Chapter 3, “Common APIs.”
L2VPN	Chapter 7, “L2VPN Provisioning.”
MPLS	Chapter 6, “MPLS Provisioning.”
Pools	Chapter 3, “Common APIs”
QoS	Chapter 9, “QoS Provisioning.”
Reports	Chapter 5, “Monitoring APIs”
Session	Chapter 3, “Common APIs.”
SLA	Chapter 5, “Monitoring APIs”
Task	Chapter 3, “Common APIs.”
Templates	Chapter 4, “Using Templates.”
VPLS	Chapter 8, “VPLS Provisioning”

Service Model

The ISC service model uses service orders, service definitions, and service requests in the provisioning process.

- Service orders allow you to schedule a provisioning process and capture the history of the provisioning process. Service orders provide a means to group together multiple service requests. This allows ISC to download multiple configuration commands, which might be targeted to a single PE, in one step, and reduces the number of reconfigurations to a network device.

- Service requests are implemented through service orders. It is the service request that is provisioned and activated in the network. The service request defines attributes for the physical links and specifies the service policy to use. A service policy is defined in service definitions.
- Service definitions define the service policy. When you define a service policy, you can also set an additional attribute (**editable=true**) for policy properties. This allows the service request creator to override certain policy attributes. Service orders and service requests use service definitions to define common data used during the provisioning process.

Service Orders/Service Requests

ISC services can be defined as either end-user services or infrastructure services. An end-user service is available to an end user (individual or organization) for which a service provider generates revenue. An infrastructure service is required to be in place before an end-user service can be offered, and the infrastructure service cannot by itself be offered as an end-user service. The ISC API supports both types of these services using service orders.

A service order allows a service provider to track the creation, modification, or deletion of all service requests implemented using ISC.

Service orders can be created to:

- Specify one or more service requests, for batch operations.
- Modify an existing service request.
- Specify the order of implementation for service requests.
- Implement many disjoint operations. One service order can modify an MPLS service request and perform other operations at the same time.
- Perform related operations. One service order can create an organization, a service definition, and a Cisco router.

Service Order Life Cycle

The following is the typical life cycle of a service order:

- The service order is created and the service request is specified.
- ISC receives the service request.
- The service request is implemented based on the due date. If the service order or any service requests within the service order has a due date in the future, it is placed in the schedule queue.
- The service request is executed at the appropriate time.
- A response message is generated to indicate if the service request has successfully deployed.

Modifying a Service Request

To make changes to a service request that has already been deployed, you must create a new service order that modifies the existing service request. The new service order becomes the *Active* service order. The previous service order becomes a historical record for the active service. Similarly, to delete a service request, you must create a new service order to decommission the existing service request.

Note When you modify a service request that has templates, or before you can decommission a service request that has templates, you must first remove the template information from the service request. See the "Removing Template Configurations" section on page 4-12 for more information.

Service Order Example

A service order XML request has a header with general information and is followed by one or more service requests and a service definition. Some of the header information can be used across multiple service requests. The service request contains service specific information, but it can also be used to override the global parameters defined in the service order header.

The following example shows a service order XML request with header information and the contained service request.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns0="http://www.cisco.com/cim-cx/2.0"
  xmlns:ns1="urn:CIM">
  <soapenv:Header>
     <!-- WaitTimeout has a default set in system properties.-->
     <ns0:message id="87855" timestamp="2002-12-13T14:55:38.885Z"
              Wait="false" WaitTimeout="60" sessiontoken="p36bttjwy1"/>
  </soapenv:Header>
  <soapenv:Body>
   <ns1:performBatchOperation>
    <actions xsi:type="ns1:CIMActionList"
             soapenc:arrayType="ns1:CIMAction[]">
     <action>
      <actionName xsi:type="xsd:string">createInstance</actionName>
      <objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">ServiceOrder</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">ServiceName</name>
         <value xsi:type="xsd:string">ServiceOrder-ConfigAudit</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">CarrierId</name>
         <value xsi:type="xsd:string">5</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">DesiredDueDate</name>
         <value xsi:type="xsd:dateTime">2002-12-14T14:55:38.885Z</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">NumberOfRequests</name>
         <value xsi:type="xsd:string">1</value>
        </item>
       </properties>
      </objectPath>
     </action>
     <action>
      <actionName xsi:type="xsd:string">createInstance</actionName>
      <objectPath xsi:type="ns1:CIMObjectPath">
```

```
<className xsi:type="xsd:string">ServiceRequest</className>
<properties xsi:type="ns1:CIMPropertyList"
             soapenc:arrayType="ns1:CIMProperty[]">
 <item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">RequestName</name>
  <value xsi:type="xsd:string">CONFIG-AUDIT-TASK</value>
 </item>
```

Names and Locator IDs

When you create a service order or a service request, you must specify a service name. Use these names to facilitate queries.

Some example service names are:

- For service orders—AcmeServiceOrder1, L2PN-ATM-SO.
- For service requests—MPLSServiceRequest, L2VPN-FrameRelaySR.

When you submit a service order XML request, ISC returns a **LocatorId** in the XML response. This locator ID is associated with the service order or request **Name**. The locator ID is unique across all service request types. MPLS, or TemplateData are examples of service request types.

Tip Make a record of the locator ID or service name for all service orders and service requests. The locator ID is required to view a service order, to perform a service order task (configuration audit or functional audit), and for all subsequent requests related to the service order or service request.

Responses to service orders and service requests also contain a **TaskLocatorId**, which can be used to retrieve log information for failed service requests. For more information, see the "Viewing Task Logs" section on page 5-23.

Service Definitions

A service definition defines a service policy and its characteristics. Service definitions can be specified in a service request, but they are not required. Use service definitions to create configuration parameters that can be used by multiple services.

ISC supports service definitions for each of the service types (MPLS, L2VPN), for QoS link services, and for templates.

Refer to the appropriate chapter on service provisioning for more information. For more information on template service definitions, see "Template Service Definitions" section on page 4-3.

API Error Messages

The API is a request/response system. The input messages are processed for errors, and any errors are reported back to the API client as part of the XML response. Errors are formatted into a standard encoding scheme. The encoding scheme is a set of three attributes as shown by the following schema:

```
<xs:element name="error">
              <xs:complexType>
                     <xs:sequence>
                            <xs:element ref="code"/>
                            <xs:element ref="description"/>
                            <xs:element ref="detail"/>
```

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

The error defines a fundamental error that prevented a method or operation from executing normally.

- The **code** attribute contains a numerical status code indicating the nature of the error.
- The **description** attribute provides a human-readable description of the error.
- The **detail** attribute, when populated, provides additional clarifications.

Note Valid status codes and descriptions are defined in *Cisco IP Solution Center System Error Messages*.

An error can be caused by more than one ISC component. For example, the code and description might refer to the API error, and the details might have information regarding an error in another component.

ISC processes error messages according to the context (create, delete, modify) in which the error was found and the class in which the error was realized. For this reason, the API returns both the operation and classname in the XML response.

The noteworthy text in the following message is indicated in bold:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmls
oap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance" xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message id="87855" sessiontoken="040833748184101892B5C1130544B053"
timestamp="2003-10-29T17:30:23.199
Z" />
  </soapenv:Header>
  <soapenv:Body>
    <ns1:createInstanceResponse>
      <returns xsi:type="ns1:CIMReturnList" soapenc:arrayType="ns1:CIMReturn[]">
        <objectPath xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">VPNServicesModule</className>
          <errors xsi:type="ns1:CIMErrorList" soapenc:arrayType="ns1:CIMError[]">
            <error xsi:type="ns1:CIMError">
              <code xsi:type="xsd:int">1104</code>
             <description xsi:type="xsd:string">Unable to find object (Device) with value
(CatIOS). Referenced object does not exist.</description>
              <detail xsi:type="xsd:string">For input string: "CatIOS"</detail>
            </error>
          </errors>
        </objectPath>
      </returns>
    </ns1:createInstanceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

In this example:

- The **createInstanceResponse** indicates the error is associated with a *create* operation.
- The class that the create operation is being performed on is **VPNServicesModule**.
- The error code, **1104**, is used to find the message in the *Cisco IP Solution Center System Error Messages*. The message is a concatenation of code and description. Hence, it is **1104- Unable to find object (Device) with value (CatIOS). Referenced object does not exist**.

Note The block call <**errors**> are used when there is more than one <**error**> for one response.

The following sample API error message shows a **createInstanceResponse** for a **ServiceRequest**.

```
<actionName xsi:type="xsd:string">createInstanceResponse</actionName>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">ServiceRequest</className>
            <errors xsi:type="ns1:CIMErrorList" soapenc:arrayType="ns1:CIMError[]">
              <error xsi:type="ns1:CIMError">
                <detail xsi:type="xsd:string">ORA-00060: deadlock detected while waiting
for resource
</detail>
                <description xsi:type="xsd:string">22 : SQL Exception  while updating
com.cisco.vpnsc.repository.mpls.RepMplsSR</description>
                <code xsi:type="xsd:int">22</code>
              </error>
```

In this example:

- The error is a repository error as indicated by the error code **22**.
- The description shows an error within the **MPLS SR**.
- The detail shows it as an **ORACLE deadlock**.

Error Logs

The API also provides error logs, which can be used for debugging purposes. The following example shows an NBI log in the **tmp** directory of the installation.

```
FINE: getErrorMsgByName(2029)
Oct 29, 2003 12:15:57 PM com.cisco.vpnsc.repository.common.RepVpnscLogger severe
SEVERE: [NbiException.processException:
com.sybase.jdbc2.jdbc.SybSQLException: ASA Error -196: Index 'MGMT_ADDR_CR' for table
'CISCO_ROUTER' would not be unique
        at com.sybase.jdbc2.tds.Tds.processEed(Tds.java:2538)
        at com.sybase.jdbc2.tds.Tds.nextResult(Tds.java:1922)
        at com.sybase.jdbc2.jdbc.ResultGetter.nextResult(ResultGetter.java:69)
        at com.sybase.jdbc2.jdbc.SybStatement.nextResult(SybStatement.java:201)
        at com.sybase.jdbc2.jdbc.SybStatement.nextResult(SybStatement.java:182)
                                .....
```

In this example:

- In the call to **getErrorMsgByName**, with argument **2029**; 2029 is the error code.
- The **ASA Error** (from SYBASE) is shown in the detail field.

The stack trace information can be used to assist the Cisco Technical Assistance Center (TAC).

Getting Started

This chapter describes how to begin service provisioning with the Cisco IP Solution Center (ISC) application program interface (API) and includes API-related installation notes, verifying the status of the API, and the typical work flow steps.

This chapter contains the following sections:

Installation Notes

All components needed for the API are included with the installation of ISC. The API servlet has no additional startup or shutdown requirements than normal ISC, as the API servlet is embedded in the Tomcat engine. The API is a common component and is aware of blade-specific data in any given installation.

The end-user interface for the ISC API interface is XML-encoded messages. A java client library and sample messages are shipped as part of the ISC product.

Refer to the *Cisco IP Solution Center Installation Guide, 4.1* for more information.

Checking the API Status

You can verify the status of the API from the API itself, or from the GUI.

The GUI and the API are both on the same port, 8030 (port=8443 for HTTPS). Both clients run from the Tomcat server. Normally, if the GUI client is started, the API client is also started.

Checking from the API

To view the status of the API, you can:

- View the status of the watchdog client. Enter the following command:

```
wdclient status
```

The following is an example output for the **wdclient status** command:

```
$ wdclient status

Status on machine : valmont.cisco.com
Name                 State           Gen   Exec Time                 PID    Succ
ess  Missed
cornerstonebridge    started         1     Oct 14 09:56:32 MDT       2622   72
     0
worker               started         1     Oct 14 09:56:32 MDT       2615   72
     0
dispatcher           started         1     Oct 14 09:56:32 MDT       2621   72
     0
lockmanager          started         1     Oct 14 09:56:32 MDT       2616   72
     0
nspoller             started         1     Oct 14 09:56:27 MDT              73
     0
scheduler            started         1     Oct 14 09:57:38 MDT       2639   71
     0
httpd                started         1     Oct 14 09:56:32 MDT       2617   71
     0
dbpoller             started         1     Oct 14 09:56:27 MDT              73
     0
cnsserver            started         1     Oct 14 09:56:33 MDT       2623   73
     0
```

The httpd is the Tomcat server for the GUI and API. If the API is running, the httpd server status is *started*.

- Attempt to log into the API.

  ```
  runNbi x Login.xml
  ```

  – If the API is not running, you receive a message:

  ```
  Failed to start client, Connection Refused.
  ```

  – If the API is running, you receive a session token.

Checking from the GUI

To check the status of the API from the GUI:

Step 1 From the Administration tab, go to **Control Center** > **Hosts**.

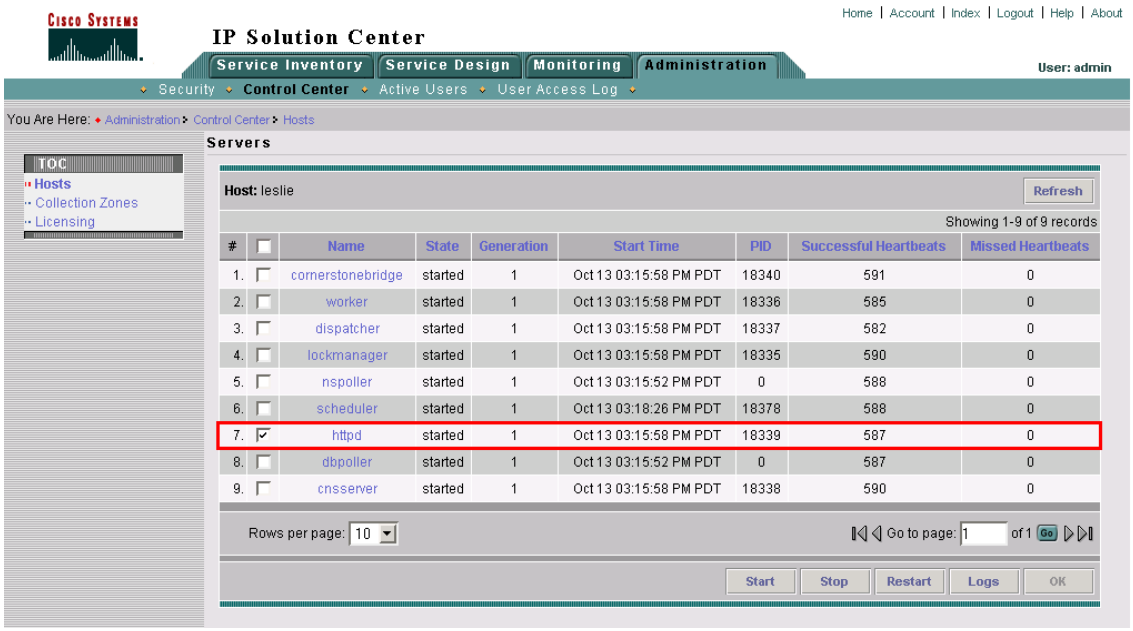
Step 2 Choose your server.

Step 3 Click **Servers**.

Step 4 Check the state of the httpd server. The httpd server is the http Tomcat server for the API client. If the API is running, it is in the *Started* state.

Figure 2-1 shows the status of the httpd server.

Figure 2-1 Server Status Window in the GUI



Step 5 To view more details, select the httpd server and click **Logs**.

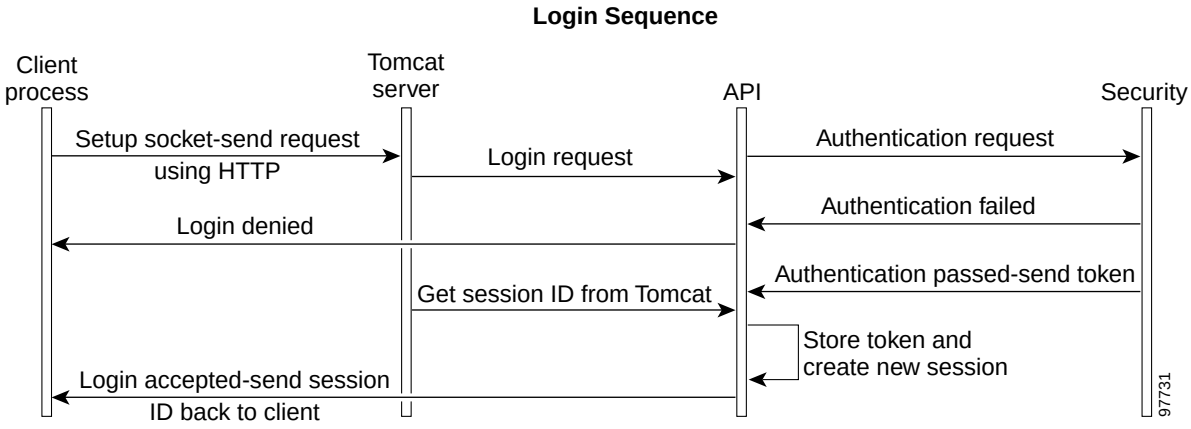
Logging In

ISC uses Cisco role-based access control (RBAC) for user login and logoff. These user roles and permissions are set up using the GUI.

When a user sends a login request, the login is validated against the RBAC processor. If the validation is successful, an RBAC security token (session token) is maintained internally and also returned in the XML response as the **SessionId**. Each subsequent request requires the security token. ISC uses this security token to restrict access to ISC data. The session token is generated by ISC from the Tomcat session ID.

Figure 2-2 shows the user login sequence.

Figure 2-2 User Login Sequence



The following is an example of an XML response to a **createSession** (Login) operation. The security token is indicated in **bold**.

```
<soapenv:Body>
    <ns1:createSessionResponse>
      <returns xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">SessionId</name>
          <value xsi:type="xsd:string">F322D1A95424C095B58083C5C3A45633</value>
        </item>
      </returns>
    </ns1:createSessionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The security token in the XML response must be used in the header message for all subsequent requests to the server.

Work Flow

This section describes the typical order of operations when using the API for service provisioning. The fundamental steps include:

- Populating the Repository, page 2-5
- Collecting Device Configurations, page 2-10
- Creating the Service Policy, page 2-11
- Creating the Service Order/Service Request, page 2-11
- Performing a Configuration Audit, page 2-12

Most end-to-end provisioning services require most or all of these operations.

Populating the Repository

Every network element that ISC manages must be defined as a device in the system. An element is any device that ISC can collect configuration information from. Additionally, you must define device roles, groups, resource pools, and the topology of the network. Each of these items represents inventory in the ISC database.

ISC offers several options for populating the inventory in the repository.

- Inventory Manager—The Inventory Manager (IM) is a stand-alone Java Application on a client machine that allows you to import and administer network-specific data into the ISC database. You must access the IM from the ISC GUI. Use the IM for:
  - Autodiscovery—A IM feature that automatically discovers the state of the service provider's network, including physical connections, encapsulation types, and routing information.
  - Importing configuration files
- GUI—Use the Inventory and Connection Manager in the GUI to add each object.

  For more information on IM and the ISC GUI, refer to the *Cisco IP Solution Center Infrastructure Reference*.
- API—Use the API to create each object and then collect the configurations from each device or device group.

This section describes the basic steps to populate the ISC repository using the API.

Note Use the XML example CreateInventory.xml for populating the database in bulk.

Populating the repository typically includes the following operations:

- Creating the inventory
- Defining provider and customer relationships
- Assigning route aggregation
- Defining access domains and VPNs
- Creating the physical topology

Creating Inventory

To provision most ISC services, the following steps for creating inventory are required:

- Create devices
- Create device groups
- Create providers and regions
- Assign PE devices to access domains
- Create customers and sites, and assign CPEs to them.

See the "Inventory" section on page 3-1 for a complete list of inventory items that can be created using ISC.

Defining the Provider and Customer Relationship

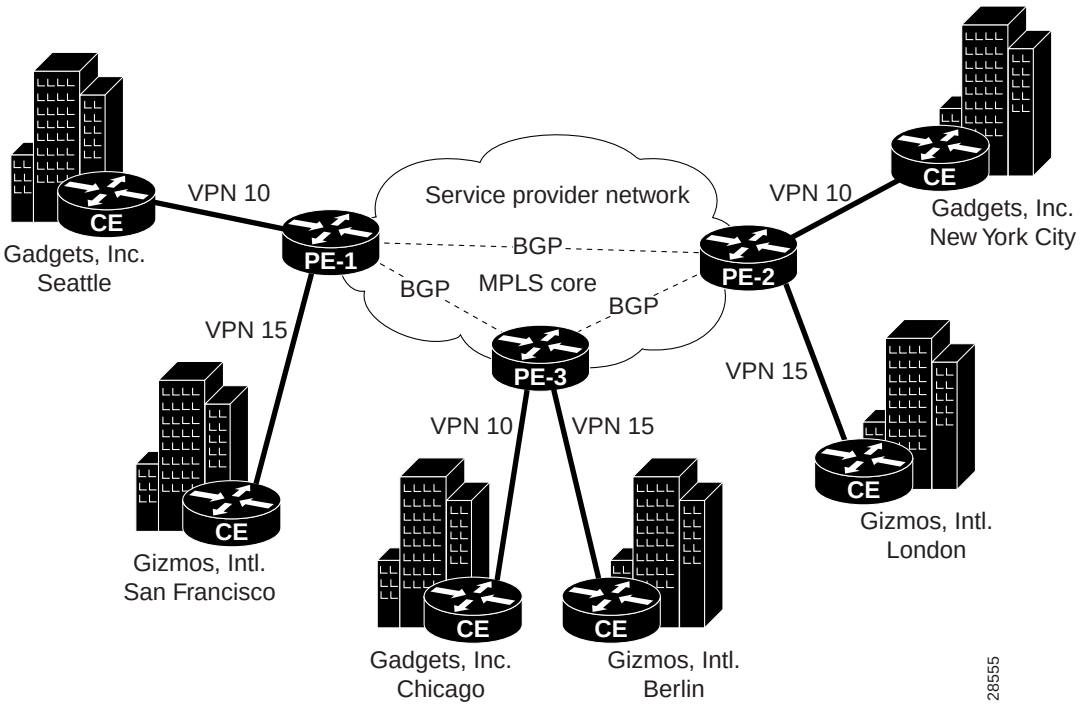
A service provider network architecture contains access routers, distributed routers, and core routers or ATM switches. Access routers terminate customer connections at the edge of the network. ISC provisioning is configured on the access circuit that involves the access router (provider edge devices or PEs) in the service provider network, and the customer premise equipment (CPE) in the customer network.

Provider

The provider administrative domain is the administrative domain of an ISP with one BGP autonomous system (AS) number. The network owned by the provider administrative domain (PAD) is called the backbone network. If an ISP has two AS numbers, you must define it as two PADs.

A service provider supplies VPN services to multiple customers. Figure 2-3 shows two different customers, each with a single VPN.

Figure 2-3 Provider View of the Network



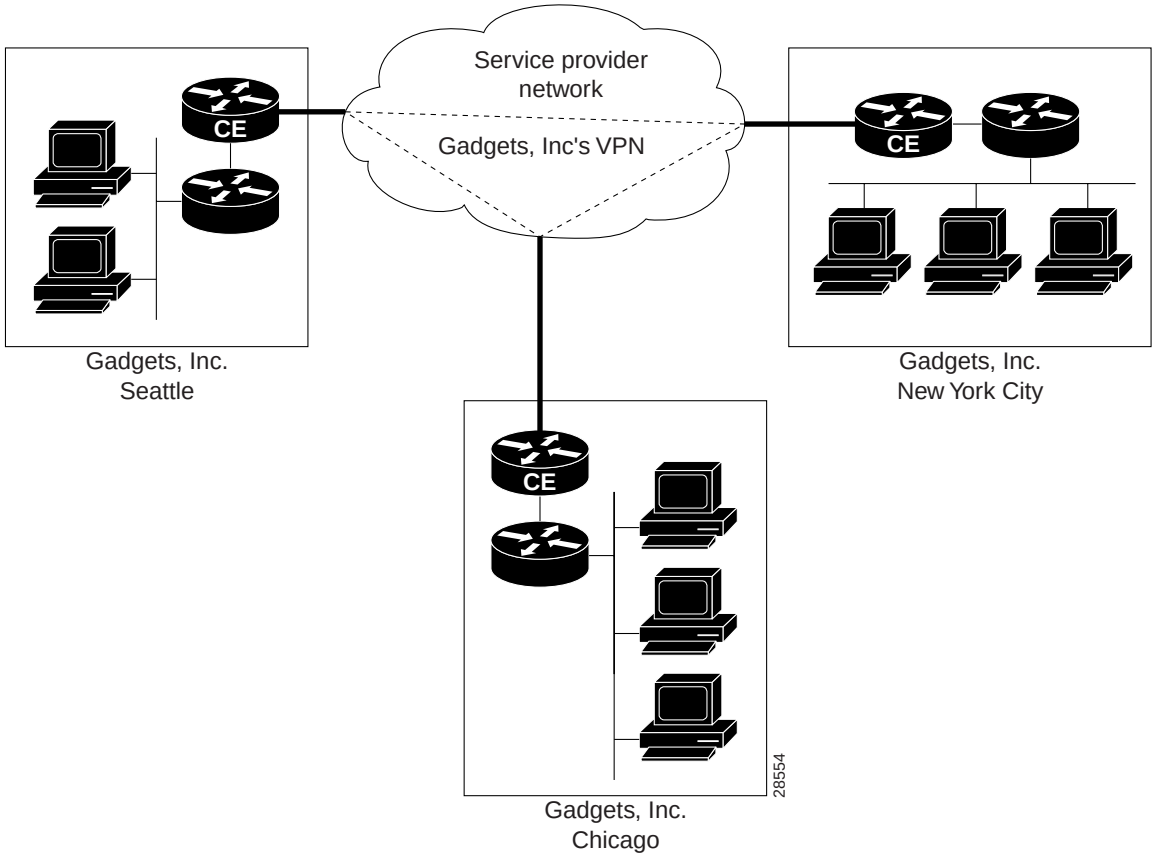
To create a **Provider** object, you need:

- Provider name (**Name**)
- Autonomous system number (**AsNumber**)

Customer

Customers have internal routers that communicate with their own customer edge devices (CEs) from one site to another through a VPN, which is managed by the service provider. See Figure 2-4.

Figure 2-4 Customer View of the Network



By using VPNs, the customers experience direct communication to each site as though they had their own private network, even though their traffic is traversing a public network.

To create a customer object, you need:

- A customer (**Organization**)

Region

A region is group of PE devices within a single BGP AS or PAD. Each provider can contain multiple region objects.

To create a Region object, you need:

- Region name (**Name**)
- Provider domain for this region (**Provider**)

Site

A customer site is a set of IP systems with mutual IP connectivity between them, without the use of a VPN. A customer site can belong to only one customer, and can contain one or more CPEs, for load balancing.

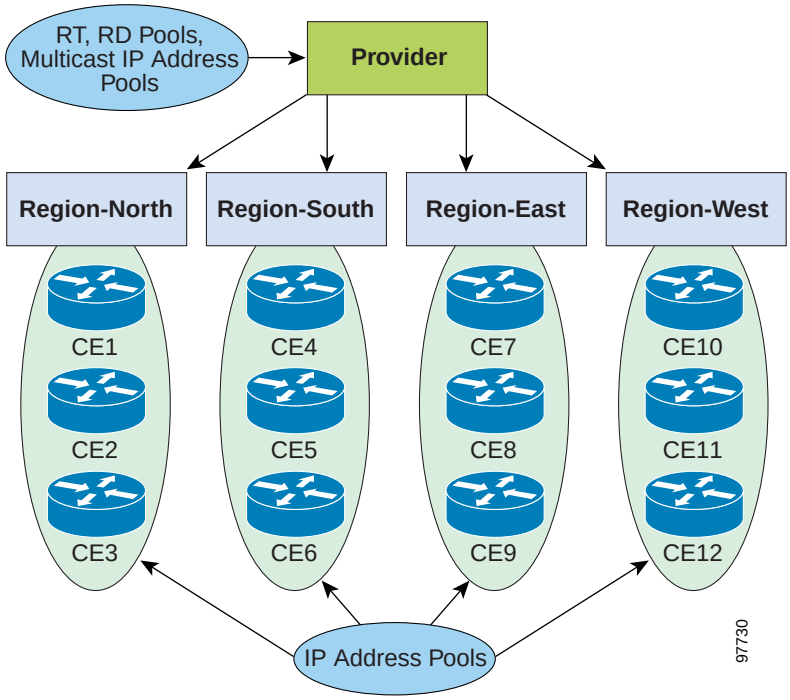
To create a **Site** object, you need:

- Site name (**Name**)
- Customer for this site (**Organization**)

Assigning Route Aggregation

Resource pools are assigned per provider and are used for route aggregation within regions. Resource pools include the route distinguisher (RD), route target (RT), and address pools. See Figure 2-5.

Figure 2-5 Resource Pools



Resource pools allow you to manage various pool types, and to associate a pool with any service model object in the ISC repository. Pools help to automate service deployment.

See the "Resource Pools" section on page 3-3 for a complete list of the supported resource pools.

Defining Access Domains and VPNs

Access domains characterize the physical relationship of the devices between the provider network and the CE devices. VPNs characterize the routing relationship between the CEs through the provider network.

Access Domains

An access domain is the Layer 2 Ethernet switching domain that is attached to the PE. All switches attached to the PE-POP belong to the same access domain. You can associate multiple VLAN pools to a single access domain. You can assign two PEs to an access domain for redundancy.

To create an **AccessDomain** object, you need:

- Provider name (**Provider**)
- The VLAN pool reserved for this domain (**ReservedVlanPool**)—The range of the VLAN pool is specified with **Start** and **Size**.

Virtual Private Networks

A virtual private network (VPN) is a collection of customer sites that share the same routing table. A VPN can consist of sites that are all from the same enterprise (intranet), or from different enterprises (extranet). VPNs can consist of sites that all attach to the same service provider backbone or to different service provider backbones.

The path between two sites in a VPN, and the characteristics of that path, can also be determined (in whole or in part) by the service policy (service definition). See Figure 2-4 for an example of a VPN between customer sites.

To create a VPN, you need:

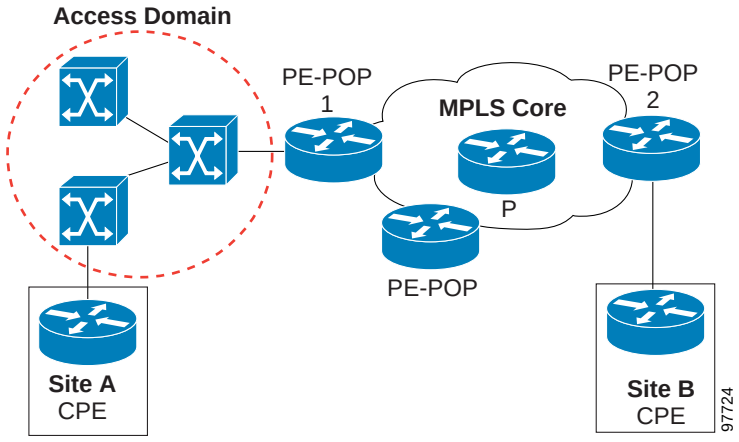
- VPN name (**Name**)
- Customer (**Organization**)
- CE routing community (**CERC**)—For MPLS VPNs only.

Creating Physical Topology

If you use a manual method for populating the repository (any method except Autodiscovery), you must also create the physical network topology. Named physical circuits (NPCs) are the physical connections between network elements and their PEs. You must create a connection from each network device in an access domain. NPCs are required for L2VPN and MPLS provisioning.

Figure 2-6 shows a network example with a distributed PE scenario. The CPE is connected to the PE through an access domain, which contains a layer 2 switching environment.

Figure 2-6 Sample L2VPN Network



To create the NPCs for this network, first specify the link from the CPE at Site A to the PE-POP 1, and then one physical link for each intermediate switch between the CPE and PE-POP.

For each **NPC** object, you need:

- Source device (**SrcDevice**)
- Destination device (**DestDevice**)
- Source device interface name (**SrcIfName**)
- Destination device interface name (**DestIfName**)

See the "Topology" section on page 3-5 for more information on the APIs available for defining network topology.

Collecting Device Configurations

A device configuration collection is a task. This task uploads the current configuration from the device to the ISC database. The collection task is executed through a service request or a service order.

The following information is required for each device that you collect configurations from:

- General device information
- Login and Password information
- Device and configuration access information:
  - Device access protocol (Telnet, SSH, CNS)
  - SNMP Version
  - SNMP V1/V2/V3 information
  - Terminal Server Information

Collection tasks can be executed immediately or they can be scheduled. See the "Collection" section on page 3-10 for more information.

Creating the Service Policy

A service policy is defined in a service definition. The service definition defines the service policy and policy characteristics. You can set an additional attribute (**editable=true**) for policy properties in the service definition to allow the service request creator to override the policy attributes. Service orders and service requests use service definitions to define common data to be used during the provisioning process.

Use service definitions to create configuration parameters that can be used by multiple services. ISC supports service policies for each of the service types (MPLS, L2VPN), and for QoS link services.

To create a service definition for most services, you need:

- Service definition name (**Name**)
- Service definition type (**Mpls**, **L2Vpn**, **QoS**)
- Service definition details (**ServiceDefinitionDetails**)—The service definitions details specify the policy information.

Refer to the appropriate chapter on provisioning for more information on creating service policies for:

- Template Service Definitions
- MPLS Service Definitions
- L2VPN Service Definitions
- QoS Service Definitions

Creating the Service Order/Service Request

Service orders allow you to schedule a provisioning process and capture its history. Service orders are used to implement service requests. It is the service request that is provisioned and activated in the network. The service request defines attributes for the physical links and specifies the service policy (defined in a service definition) to use.

Use service orders to:

- Specify and implement one or more service requests, for batch operations
- Modify an existing service request
- Specify the order of implementation for service requests
- Initiate a task

Service request deployment is scheduled based on the service order or service request due date. If the service order or any service requests within the service order has a due date in the future, it is placed in a schedule queue.

To schedule a service order, you need:

- Service order name (**ServiceName**)
- The number of requests (**NumberOfRequests**)
- The service request to implement (**ServiceRequest**)—The service request details specify the attribute and service policy information. You can specify one or more service requests.

Service Order Response

The XML response to a service order contains its own **LocatorId** (required) and **ServiceName** (optional). If the service order is created to implement a service request, the service order response also contains the response to the service request, which includes the **LocatorId** (required) and the **ServiceRequestName** (optional).

If the service order is created to initiate a task (collect configurations, perform functional audits), the service order XML response also contains data. Use the locator ID for subsequent requests relating to the service order or service request.

You can view any service order record using the **LocatorId** attribute. The response to a view request contains a **TaskLocatorId** attribute, which is used to track the status of any task created in conjunction with the service order. The **TaskLocatorId** attribute also allows you to view the task logs. See the "Viewing Task Logs" section on page 5-23 for more information.

Refer to the appropriate chapter on provisioning for more information on creating service orders.

Performing a Configuration Audit

You can perform a configuration audit as part of a service request deployment or as a separate operation. During a configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. A configuration audit also verifies that there were no errors during service deployment.

A configuration audit is considered a task in the API and is executed with a service order. To schedule a configuration audit you need to specify:

- The type of audit you want to perform in the service request details (**SubType=CONFIG_AUDIT**)
- Locator ID of the service request that was used to deploy the configuration to audit (**LocatorId**).

Note If the configuration audit is part of a service request deployment, you do not need to perform an audit as a separate operation.

Configuration Audits are described in the "Tasks" section on page 3-8.

Common APIs

This chapter describes the Cisco IP Solution Center (ISC) APIs for operations that are common to all ISC services. These operations include; creating inventory in the ISC respository, session login, auditing, deployment and collection tasks, database event notifications, and general purpose XML requests.

This chapter contains the following sections:

Inventory

You can create, modify, delete, or view any inventory object in the ISC repository. Use the following API operations to:

- Create an object—createInstance
- Modify an object—modifyInstance (Resource pools do not support the *Modify* operation.)
- Delete an object—deleteInstance
- View an object—enumerateInstances

Note When you send a view request (**enumerateInstances**) for an object, the response returns the *create* and *modify* date information.

Inventory APIs are divided into the following categories:

- Devices—Physical devices in the network
- Resource Pools—IP address pools, VLAN pools, route target, route distinguisher
- Topology—CE routing communities (CERCs), virtual private networks (VPNs), and named physical circuits (NPCs)
- Groupings—Access domains, device groups, collection zones.

Note XML examples for all APIs are located at the following URL: http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Devices

The inventory APIs for devices allow you to define a physical device in the ISC repository. Every network element that ISC manages must be defined as a device in the system. An element is any device from which ISC can collect configuration information.

Table 3-1 shows the devices available as inventory objects.

Table 3-1 Inventory Objects—Devices

className	Required Parameters	XML Examples
AAAServer	• Number of Retries • Address • AuthServerType= – RADIUS – NTDOMAIN – SDI – TACACS+ • Role= – AUTHENTICATION – ACCOUNTING – BOTH	• CreateAAServer.xml • CreateAAServerNTDOMAIN.xml • CreateAAServerRADIUS.xml • CreateAAServerSDI.xml • CreateAAServerTACACS.xml
CatOs (Cat OS operating system)	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreateCat.xml
CatIOS (Cisco IOS operating system)	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreateCatIOS.xml
CiscoRouter	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreateCiscoRouter.xml

Table 3-1 Inventory Objects—Devices (continued)

className	Required Parameters	XML Examples
Cpe	• Site • Device • Name	CreateCpe.xml
PE	• Device • Region	CreatePE.xml
PIX	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreatePIX.xml
TerminalServer	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreateTerminalServer.xml
VPNServicesModule	• Parent • SlotNumber	CreateVPNSM.xml
IE2100	One or more of the following: • IPAddress • HostName • DomainName	CreateIE2100.xml
NonCiscoDevice	One or more of the following: • ManagementIPAddress • HostName • DomainName	CreateNonCiscoDevice.xml

Resource Pools

Resource pools allow you to manage various pool types and to associate a pool with any service model object in the ISC repository. Pools help to automate service deployment.

ISC supports the following resource pools:

- IP address pools—Defined and assigned to regions.
- Multicast pools—Used for multicast MPLS VPNs.
- Route Distinguisher (RD) pool—The IP subnets advertised by the CPE devices to the PE devices are augmented with a 64-bit prefix, which is the route distinguisher.
- Route Target (RT) pool—The MPLS mechanism that informs PEs which routes to insert into the appropriate VPN routing and forwarding table (VRF). Every VPN route is tagged with one or more route targets when it is exported from a VRF and offered to other VRFs.

- Site-of-origin pool—The pool of values for the site-of-origin attribute. The site-of-origin attribute prevents routing loops when a site has multiple connections to the MPLS VPN backbone.
- VLAN ID pool—VLAN ID pools are attached to an access domain. To have ISC automatically assign VLANs to end-to-end wire links (for L2VPN provisioning), you can specify the **AutoPickVlanId** option.
- VC ID pool—A 32-bit identifier that is shared between the two PEs. The VC ID is a globally unique value.

Note The *Modify* operation is not supported with resource pools.

Table 3-2 shows the resource pools available as inventory objects.

Table 3-2 Inventory Objects—Resource Pools

className	Required Parameters	XML Examples
IPAddressPool	• IPAddressPool • SubnetMask • AssocClassType= – Region – VPN	CreateIPAddressPool.xml
MulticastAddrPool	• MulticastAddress	CreateMulticastAddrPool.xml
RouteDistinguisher	• Start • Size • AssocClassType=Provider • AssocClassId	CreateRouteDistinguisher.xml
RouteTarget	• Start • Size • AssocClassType=Provider • AssocClassId	CreateRouteTarget.xml
SiteOfOrigin	• Start • Size • AssocClassType=Provider • AssocClassId	CreateSiteOfOrigin.xml
VcIdPool	• Start • Size	CreateVcIdPool.xml
VlanIdPool	• Start • Size • AssocClassType=Provider • AssocClassId	CreateVlanIdPool.xml

Topology

In complex topologies, it is sometimes necessary to further define the connectivity between CE and PE devices into groups, such as CE routing communities (CERCs), virtual private networks (VPNs), and named physical circuits (NPCs).

Tip Use the Topology tool in the GUI to check CERC memberships and the resulting VPNs.

Table 3-3 shows the topology elements available as ISC inventory objects.

Table 3-3 Inventory Objects—Topology

className	Required Parameters	XML Examples
CERC	• Provider	• CreateCERC.xml **Note** The CreateCERC.xml appends to any existing CERC. To create a new CERC, first run a modify CERC to remove the existing CERC attributes. • CreateVPN_DefaultCERC.xml
VPN	• Name • Organization	• CreateVPN.xml
NamedPhysicalCircuit	• PhysicalLink – SrcDevice – DestDevice – SrcIfName – DestIfName	• CreateNamedPhysicalCircuit.xml
NamedPhysicalCircutRing	• LocatorId	• CreateNamedPhysicalCircuitRing.xml • CreateNamed PhysicalCircuitRingExisting.xml

Tip Use **ViewVRF.xml** to view the VRF routing tables.

Groupings

ISC is designed to provision a large number of devices through its distributed architecture. Groupings are provided to simplify management tasks of devices for administrative or geographical reasons or for scalability.

- Access domain—The access domain object is associated with the provider access domain (PAD). Each PE-POP is assigned to an access domain.
- Device groups—Devices that are organized for collection and management purposes.
- Organization (Customer)— A requestor of VPN services from an Internet service provider (ISP).

- Site—A set of IP systems with mutual IP connectivity between them without the use of a VPN.
- Provider— An enterprise that provides internet services for customers.
- Region—A group of PE devices within a single border gateway protocol autonomous system (BGP AS).
- Collection zones—Collection servers are used to off load the work of the master server when the number of devices in the network increases. Network devices are associated with these collection servers using collection zones.
- Network objects—A network object is a group of IP addresses to use for traffic classification in a QoS policy instead of specifying a single IP address.

Table 3-4 shows the groupings as inventory objects.

Table 3-4 Inventory Objects—Groupings

className	Required Parameters	XML Examples
AccessDomain	• Provider • ReservedVlanPool – Start – Size – Managed	CreateAccessDomain.xml
DeviceGroup	• Name • Devices	CreateDeviceGroup.xml
Organization	• Name • ContactInfo • SiteOfOrigin=enabled (use for multiple connections to the MPLS backbone)	CreateOrganization.xml
Site	• Name • Customer	CreateSite.xml
Provider	• Name • AsNumber	CreateProvider.xml
Region	• Name • Provider	CreateRegion.xml

Table 3-4 Inventory Objects—Groupings (continued)

className	Required Parameters	XML Examples
CollectionZone	• Name • VPNSCHost	CreateCollectionZone.xml
NetworkObject	• Name • Type= – STRING – NETWORK – HOST • Values • ContainerId • FQCN (container type) – Global – Customer – Site – Cpe	CreateNetworkObject.xml

Session

Sessions are specialized API operations. ISC supports the following session operations:

- createSession—Login
- deleteSession—Logout

The first XML request sent by the client is a login request. The login request generates a session ID, which is used each time you access the system. The session ID is valid for a configurable period of time. During this time, you can make an indefinite amount of calls to the server, using the session ID for authentication. Each call to the server resets the time-to-live (ttl) time back to the original period of time. The default session time is 20 minutes.

The API license is global to the installation and is checked at the start of each session. If the API license does not exist, the session is not granted. During the session, if a user attempts to provision a service that is not licensed, an error is returned.

Table 3-5 User Login

className	Required Parameters	XML Examples
Session	• LoginName • LoginPassword	Login.xml

The following example shows the XML response to a login request. The **SessionId** is indicated in **bold**.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message id="87855" timestamp="2003-10-10T19:43:16.380Z"
sessiontoken="93E0A400406693604270C6B6A07731A4" />
  </soapenv:Header>
  <soapenv:Body>
    <ns1:createSessionResponse>
      <returns xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">SessionId</name>
          <value xsi:type="xsd:string">93E0A400406693604270C6B6A07731A4</value>
        </item>
      </returns>
    </ns1:createSessionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Tasks

Task APIs are used for auditing, deployment, and collection activities. Tasks follow the same service order/service request structure as other ISC services, where the service request is executed as part of a service order.

- For device-related tasks (Collection and SLA Collection), you must enter the device or device group.
- For service request-related tasks (Certificate Enrollment Audit, Configuration Audit, Service Deployment, and MPLS Functional Audits), you must enter the service request locator ID.

Tasks are scheduled using the **DesiredDueDate** field in the service request or service order. A task with a **DesiredDueDate** that is in the past, or within 5 minutes of the current time, is executed immediately. Tasks with due dates in the future are scheduled.

ISC uses the following guidelines for **DesiredDueDate**

- If there is no due date in the service request, the task uses the service order due date.
- If there is a date in the service request, the service request due date overrides the service order due date for that particular task.
- If there is no due date specified in the service order or the service request, the service is not deployed.

Tasks can be created, delete, and viewed using the API. However, you can only modify the **DesiredDueDate** field in a Task.

All Tasks are deployed using a service order that specifies a service request with **Type=Task**.

Viewing a Task

To obtain information about task service requests, run a view of the task (ViewTask.xml), using the **TaskLocatorId**.

Use this process to retrieve the **TaskLocatorId**:

Step 1 Record the service order **LocatorId** that is returned in the XML response.

When you submit a service order XML request for any task, ISC returns a **LocatorId** in the XML response. This locator ID is associated with the service order or request **Name**.

Step 2 Run a view of the service order using ViewServiceOrder.xml, and the **LocatorId** from Step 1.

Step 3 Record the **TaskLocatorId** that is returned in the XML response.

Step 4 Run a view of the Task using ViewTask.xml. Specify **className=PersistentTask** and use the **TaskLocatorId** from Step 3.

The API supports the following tasks:

- Certificate Enrollment Audit
- Collection
- Service Decommission
- Configuration Audit
- Service Deployment
- MPLS Functional Audit
- MPLS Functional Audit
- SLA Collection

Certificate Enrollment Audit

A certificate enrollment audit is performed on devices to verify the certificates issued to them by a certificate authority (CA).

A certificate enrollment audit does the following:

- Verifies the certificates issued by the CA to a device or device groups.
- Confirms the presence of the root certificate and device certificate for the certificate chain of the specified trust point.
- Returns a summary that indicates the certificate enrollment status and expiration status on the audited devices.

Table 3-6 Certificate Enrollment Task

className	Required Parameters	XML Examples
ServiceRequestDetails	• SubType= CERT_ENROLLMENTAUDIT • Device (or DeviceGroup) • TrustedPort • IncludeRootCert	CreateTaskServiceOrderCertAudit.xml

Collection

A collection operation collects configuration information from network devices and serves the following purposes:

- It loads the current configuration information for the devices that populate many of the configuration cells.
- It verifies reachability and passwords for the devices from which it collects configuration files.

Table 3-7 Collection Task

className	Required Parameters	XML Examples
ServiceRequestDetails	• SubType=COLLECTION • Device (or DeviceGroup) **Note** You must select at least one device or device group. • RetreiveVersion=true • RetreiveDeviceInterface=true	Create TaskServiceOrderCollection.xml

Tip Performing a device collection on your network is more accurate and efficient than manually entering individual device information.

Service Decommission

Decommissioning a service request removes a service from all devices associated with the service request.

Note To decommission a service request that includes templates, you must first negate the template information on the device. See the "Removing Template Configurations" section on page 4-12 for more information.

During the decommission process, ISC creates the necessary removal configuration to delete the service from each device and automatically audits the configuration to ensure that the service is completely removed. Once audited, the service request changes to a *Closed* state.

Table 3-8 Decommission Task

className	Required Parameters	XML Examples
ServiceRequestDetails	• SubType=DECOMMISSION • LocatorId **Note** The **LocatorId** of the service request to decommission.	CreateTask ServiceOrderDecommission.xml

If you do not want the configuration audit to occur, change the value for the **Audit** parameter. The **Audit** parameter supports these values:

- **Audit**—This is the default. A successfully deployed decommission service request is automatically audited unless this flag is changed.
- **NoAudit**—Do not perform a configuration audit when the decommission service request is deployed.
- **ForceAudit**—Perform a configuration audit even if the deployment of the decommission service request is not successful.

Configuration Audit

A configuration audit is executed as part of a service request deployment. During a configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. A configuration audit also verifies that there were no errors during service deployment.

Table 3-9 Configuration Audit Task

className	Required Parameters	XML Examples
ServiceRequestDetails	• SubType=CONFIG_AUDIT • LocatorId **Note** The **LocatorId** of the service request used to deploy the configlets.	CreateTask ServiceOrderConfigAudit.xml

Service Deployment

A service deployment downloads the ISC-generated configlet, created for a service order, to the device configuration file.

You can specify the following options for service deployment:

- **ForceDeploy**—Takes a configlet for a service request that is already in the *Deployed* state and downloads it to the network device. Use **ForceDeploy** when a device configuration is lost or when you replace or change equipment.

Table 3-10 Deployment Task

className	Required Parameters	XML Examples
ServiceRequestDetails	• SubType=DEPLOYMENT • Provision • LocatorId **Note** The **LocatorId** of the service request used to deploy the configlets.	Create TaskServiceOrderDeployment.xml

MPLS Functional Audit

An MPLS functional audit verifies that the links in a service request or VPN are working correctly. The audit checks the routes to remote CPE devices in the VRF route tables on the PE devices.

Table 3-11 MPLS Functional Audit Task

className	Required Parameters	XML Example
ServiceRequestDetails	• SubType= MPLS_FUNCTIONALAUDIT • LocatorId **Note** The **LocatorId** of the service request used to deploy the configlets. • VPN • MPLSMirrorPing	CreateTask ServiceOrderMPLSFuncAudit.xml

SLA Collection

ISC uses the Cisco SA Agent MIB to monitor service level agreement (SLA) metrics. SLAs monitor the network performance by measuring response time, jitter, connect time, throughput, and packet loss. The SA Agent allows you to configure probes for performance measurements and uses a router monitor (RTRMON) MIB for access through SNMP. The MIB also supports SNMP notifications.

An SLA collection task collects all SLA data from SLA-enabled devices. ISC collects the relevant performance data and stores it persistently in the database. To view the data, you must execute a task to aggregate the data and run the SLA Report.

Note To collect SLA data from a device, it must have SA Agent and SNMP enabled, and SNMP parameters must already be set.

SLA collection includes performance data on Jitter (voice jitter) and the following protocols:

- Dynamic Host Configuration Protocol (DHCP)
- Domain Name System (DNS)

- File Transfer Protocol (FTP)
- Internet Control Message Protocol Echo (ICMP Echo)
- Transmission Control Protocol Connect (TCP Connect)
- User Datagram Protocol Echo (UDP Echo)
- Hypertext Transfer Protocol (HTTP)

Table 3-12 SLA Collection Task

className	Required Parameters	XML Example
ServiceRequestDetails	• SubType=SLA_COLLECTION • Device (or DeviceGroup) **Note** You must select at least one device or device group.	CreateTask ServiceOrderSLACollection.xml

General

The following general purpose XML API examples are provided with ISC:

- CreateConfigServiceOrderDownload—Creates a service order to download an inline configuration to a device.
- Create Inventory—Can be used to populate a database with sample inventory and is commonly used for testing or demonstration. The service order adds one type of each inventory element to the repository and enters the host name, domain name, management IP address, and the login and password for each device.
- CreateNPCInventory—Creates several physical links in a network, including the source and destination devices and relevant interfaces.
- CreateExecCommandServiceOrder—Creates a service order to execute a command on a device or device group. You can execute any command allowed by the device console.
- CreateServiceOrderResponse—Returns the following values for a service order:
  - **ServiceName** (for the service order)
  - **ServiceRequestName**
  - **LocatorId** (for the service request)
- CreateSLAInventory—Can be used to populate a database with sample inventory and is commonly used for testing. The service order adds the host name, domain name, and SLA data to devices in device groups.
- CreateResourceLock_Device, CreateResourceLock_Device_Batch, CreateResourceUnlock_Device, ViewResourceLock_Device—See the "Device Locking" section on page 5-21.
- DeleteServiceOrder—Deletes a service order.
- DeleteServiceOrder_purge—Deletes a service request of a specified subtype (**Type = QoS, L2VPN, Mpls**), with a given locator ID and **Force=true**.
- ViewConfiglet—View the ISC generated configlet for a specified service request. Returns the ViewConfigletResponse.
- ViewConfigletResponse—Returns the following values:

  - **ConfigletClob**
  - **LocatorId**
  - **Device**
  - **ConfigletText**—Contains the configlet.
- ViewConfiguration—View the configuration of a specified device. Returns the ViewConfigurationResponse.
- ViewConfigurationResponse—Returns the following values:
  - **LocatorId**
  - **Device**
  - **DeviceConfig**—Contains the device configuration.
- ViewCpe_propList_level—Provides a sample of a properties list and level usage. Returns the ViewCpe_propList_level_Response.
- ViewCpe_propList_level_Response—Returns the following values:
  - **LocatorId**
  - **Site**
  - **Device**
- ViewMPLSServiceRequest_propList_level—Provides a sample of a properties list and level usage. Returns the ViewMPLSServiceRequest_propList_level_Response.
- ViewMPLSServiceRequest_propList_level_Response—Returns the following values:
  - **OpType** (operation type; ADD or DELETE)
  - **LocatorId**
  - **MplsVpnLink** (and link attributes)
  - **CERCMembership**
- ViewServiceOrder—View the status of any service order, with a given service order name or locator ID. If the service order employs a task, the task locator ID is returned. The **TaskLocatorId** is required to view a task service order (**ViewTask**).

Using Templates

Cisco IP Solution Center (ISC) uses templates to generate device commands that are not supported by ISC, and to download them to a Cisco device. For example, ISC does not configure importing of root certificates. A template enables you to add this configuration to a device. A template configuration file can be either a partial or complete configuration file. The template configuration file is merged with (either appended or prepended to) the ISC configlet. The combined configlet is downloaded to the device as part of a service request or as a transient template.

Templates are defined in service definitions and can be deployed:

- Using a service order.
- Attached to a service request for another service (see the "Templates in a Service Request" section on page 4-9).

You can use the API to generate template definitions, template data, and device configlets based on the templates.

This chapter contains the following sections:

Refer to the *Cisco IP Solution Center Infrastructure Reference, 4.1* for information on the GUI Template Manager.

Template Overview

Templates consist of template definitions and template data. The template definition contains the logic and variables to be populated with template data. The template data is the configuration information to be downloaded to a device. When ISC merges the template definition's variables with the data in the template data file, a template configuration file is created. The template configuration file is downloaded to the device.

Templates can be deployed independently of other ISC functions or they can be attached to a service request.

The API supports the following types of template operations:

- Templates created from a template definition and a data file.
- Buffer templates—Template data is pulled from a data buffer instead of a data file and inserted directly into a service request (only for MPLS service requests).
- Templates integrated as part of a service request—The service request specifies the device to receive the configuration (the template definition and template data method).
- Transient templates—Transient template data is used only for the download and then discarded. It is not available for subsequent viewing (only for direct template download service requests).

Template definition files and template data files are stored in XML format. The template definition file, its data files, and all resulting template configuration files are mapped to a single directory. One template definition can contain many data files, but a template data file can be attached to only one template definition.

Tip When you generate a template configuration file using a particular template data file, the configuration filename correlates to the data filename.

To view the interaction between the template and the device, use the task logs. See the "Viewing Task Logs" section on page 5-23 for more information.

Template Definition

The template definition defines the variables that are populated with template data. It defines the actions that need to be taken for any device to which the template is attached.

The template definition specifies what data is necessary to create the template configuration file, and includes how the variable names and the data are associated.

Note The template definition in the API corresponds to the template in the GUI.

Template Data

The template data consists of name/value pairs for each variable defined in the template definition. Each template data file can be associated with only one template definition.

Template data can be created using the GUI or the API. The data can exist in a template data file, be merged with a template definition from a data buffer, or be entered as transient data directly into a service request.

Creating a template data file is a separate operation. However, if you use transient data or data buffers, this allows you to enter template data at the same time you are creating the service definition or service request.

The data file contains data for all variables in the template definition.

Note To view the configuration created using a template, without downloading the template to the device, use the ViewTemplateConfig XML request. Specify the template definition and template data, and the configuration is returned in the XML response.

Template Operations

Template definitions and template data files are specified in a service definition. The device to receive the template configuration and transient data and data buffers (if applicable) are defined in the service request as part of a service order.

The API supports these template subtypes:

- **TemplateDefinition**—The template itself, which contains the variables and logic to be populated with template data.
- **TemplateData**—The data to be merged with a template definition.
- **TemplateConfig**—The template configlet that is the result of the template definition being merged with the template data.
- **TemplateDownload**—Used to download a template configlet to a device using template data from a data file.
- **TemplateTransient**—Used to download a template configlet to a device using template data that is added directly into the XML request.

The following template operations can be executed using the API:

- For service definition subtypes:
  - **TemplateDefinition**—Create, Delete, Modify, or View
  - **TemplateData**—Create, Delete, Modify, or View
- For service request subtypes:
  - **TemplateConfig**—Create, Modify, or View
  - **TemplateDownload**—Create, Delete, Modify, or View
  - **TemplateDataTransient**—Create or View

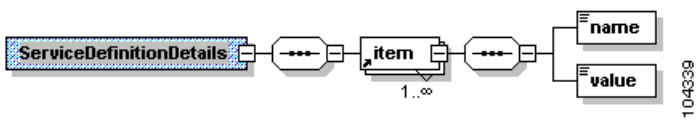
Template Service Definitions

Using service definitions to define templates enables the XML requests to be processed the same as other service policies (QoS, MPLS, and L2VPN) and allows them to be specified in service orders.

A template service definition consists of a template definition and the corresponding template data items. The template definition specifies the variable names and logic in the **BodyText** property.

Figure 4-1 shows the schema diagram for a template service definition. The *item* refers to the template definition, and the *name/value* pairs refer to the template data.

Figure 4-1 Schema Diagram for Template Service Definitions



Template Service Orders

A template is implemented using a service order. During a service request deployment, the template definition and data file are merged, and the resulting configuration is appended or prepended to the ISC-generated configlet. The combined configuration is downloaded to the device specified in the service request.

If the template is:

- Prepended—The template commands take place before the service request commands.
- Appended—The template commands take place after the service request commands.

Service orders can specify template downloads, transient data downloads, and templates specified within a service request.

To view a template service order:

- For templates that specify a data file, only the data file name is listed in the service request. Viewing the data file is a separate operation.
- For templates that specify a data buffer, the data is displayed within the service request. Template data buffers can only be viewed by viewing the service request. Use the **enumerateInstances** operation and enter the **LocatorId** of the service request that contains the template data buffers.

Provisioning Example

This section describes the required steps for using templates independent of service requests. See the "Templates in a Service Request" section on page 4-9 for information on deploying templates with service requests.

Prerequisites

ISC provides pre-populated examples to help you create a template.

- If you are using Sybase as a back-end database, you are provided with pre-populated template examples. These examples can be found on the left pane of the main Template Manager window.
- If you are using Oracle as a back-end database, you are NOT provided with pre-populated template examples. You must either create a template definition from scratch or import a template. Alternately, you can run the script **populateTemplates.sh** located in the **<install-dir>/bin** directory.

Process Summary

In this template provisioning example, the following steps are listed:

- Create a template definition file.
- Create the template data file.
- View the template configuration.
- Download the template configuration to a device.
- Delete the template data file and template definition.

Provisioning Process

This section provides an example provisioning process using XML examples. The inventory of XML examples for the ISC API can be found at the following location:

http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Step 1 Create a template definition.

Table 4-1 Create Template Definition

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=TemplateDefn • ServiceDefinitionDetails
	ServiceDefinitionDetails	Can contain one or more of the following classes, with associated variable name/value pairs as child objects: • TemplateInteger • TemplateString

XML Example:

- CreateTemplateDefnSimple.xml

Step 2 Create a template data file.

Table 4-2 Create Template Data

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=TemplateData • ServiceDefinitionDetails
	ServiceDefinitionDetails	<template data> (The name/value pairs.)

The following XML examples show how to populate a template containing one-dimensional variables or two-dimensional variables. The incoming template data must match the format of the template definition. The API validates the incoming data against the variable definition. An error is returned if they do not match.

XML Examples:

- CreateTemplateData1Dim.xml
- CreateTemplateData2Dim.xml

Step 3 View the template data file.

To view the data file, provide the full path name and filename. This is the same as the folder path name and filename in the GUI.

Table 4-3 View Template Data

Operation	className	Required Parameters
enumerateInstances	ServiceDefinition	• Name=<pathname/filename to template data file> • Type=TemplateData

XML Example:

ViewTemplateData.xml

Step 4 View the configlet generated for the template.

Table 4-4 View Configlet

Operation	className	Required Parameters
enumerateInstances	Task	• Type=TemplateConfig • TemplateDefn=<pathname to template definition> • TemplateData=<template data filename>

XML Example:

ViewTemplateConfig.xml

The following example shows a response to a ViewTemplateConfig XML request.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message id="87855" sessiontoken="F1856684E9A183F5E542890772B3D040"
timestamp="2003-09-25T21:38:07.645Z" />
  </soapenv:Header>
  <soapenv:Body>
    <ns1:enumerateInstancesResponse>
      <returns xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Configlet</name>
          <value xsi:type="xsd:string">
ip vrf $vrfName
 maximum routes 2 75
  export map To_NM_VPN
 route-target import 2:103000
 route-target import 2:103500
 route-target import 2:103020
 route-target import 2:103520
 route-target import 0
```

```
 route-target import 1
 exit
router bgp 13979
 address-family ipv4 vrf vrf1
 default-information originate
 maximum-paths eibgp 6
 bgp suppress-inactive
   neighbor 10.10.10.1 route-map set_ce_local_pref in
 neighbor 10.10.10.1 maximum-prefix 21 50
 neighbor 10.10.10.1 capability orf prefix-list receive
 neighbor 10.10.10.1 advertisement-interval 60
    exit
</value>
        </item>
      </returns>
    </ns1:enumerateInstancesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Step 5 Download the template configuration to a device.

Table 4-5 Download Template Configuration

Operation	className	Required Parameters
performBatchOperation		
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=TemplateDownload • Template definition information: – ServiceDefinition=<pathname to template definition> – ServiceDefinitionType= TemplateDefn • Template data information: – ServiceDefinition=<pathname /filename to template data file> – ServiceDefinitionType= TemplateData • ServiceRequestDetails
	ServiceRequestDetails	LogicalDevice=<name of device to receive the template configuration>

XML Example:

CreateTemplateServiceOrderDownload.xml

Step 6 Delete the template data file and the template definition file. This step is optional.

Table 4-6 Delete Template Files

Operation	className	Required Parameters
deleteInstance	ServiceDefinition	To delete the template data file: • Name=<pathname to template definition file> • Type=TemplateData To delete the template definition file: • TemplatePathname=<pathname/file name to template data file> • Type=TemplateDefn

XML Examples:

- DeleteTemplateData.xml
- DeleteTemplateDefn.xml

Transient Templates

For transient templates, the template data is not specified through a previously defined data file. The template data is entered directly into the XML request. Transient data is used only for the instance of the service order and is then discarded. The transient data is not available for subsequent service orders, and you cannot view transient data when you view the service order.

- To view the generated configlet, refer to "View the configlet generated for the template." on page 6.
- To download transient template data to a device, refer to "Download the template configuration to a device." on page 7.

For transient templates, leave out the following two properties:

- **ServiceDefinition=<pathname/filename to template data file>**
- **ServiceDefinitionType=TemplateData**

Instead, specify **SubType=TemplateDataTransient** in the **ServiceDefinition**, and enter the template data (name/value pairs) in the **ServiceDefinitionDetails**. See the following example:

```
<objectPath xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">ServiceRequest</className>
      <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
       <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">RequestName</name>
        <value xsi:type="xsd:string">MYSR-1</value>
       </item>
       <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">Type</name>
        <value xsi:type="xsd:string">TemplateDownload</value>
       </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">ServiceDefinition</name>
```

```
  <value xsi:type="xsd:string">/User/UsernameTemplate</value>
  <qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">ServiceDefintionType</name>
    <value xsi:type="xsd:string">TemplateDefn</value>
  </qualifier>
</item>
<objectPath xsi:type="ns1:CIMObjectPath">
 <className xsi:type="xsd:string">ServiceDefinition</className>
 <properties xsi:type="ns1:CIMPropertyList"
           soapenc:arrayType="ns1:CIMProperty[]">
 <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">SubType</name>
    <value xsi:type="xsd:string">TemplateDataTransient</value>
 </item>
 </properties>
 <objectPath xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
  <properties xsi:type="ns1:CIMPropertyList"
           soapenc:arrayType="ns1:CIMProperty[]">
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">username</name>
    <value xsi:type="xsd:string">user1</value>
   </item>
 </properties>
 </objectPath>
```

XML Example:

CreateTemplateServiceOrderDownloadTransient.xml

Templates in a Service Request

You can add templates to a service request. When the service order is deployed, the template configuration is downloaded to the device, along with the configuration from the service request.

Note To remove templates from a service request, see "Removing Template Configurations" section on page 4-12.

The template information in a service request template contains the **LinkTemplate** object, which defines the location of the template definition and data files, the device to receive the configuration download, and whether to prepend or append the template configuration.

Link Template

When you include templates in a service request, the template information is defined using the **LinkTemplate** object. The **LinkTemplate** contains the path to the template definition and the location of the template data.

For each service type, the **LinkTemplate** is defined in these link objects in the service request:

- MPLS—**MplsVpnLink**
- L2VPN—**ACAttr** (EndtoEndWire>AttachmentCircuit>ACAttr)
- VPLS—**VPLSLink**

Note See the appropriate chapter on service provisioning for more information on using LinkTemplate in service requests.

Data Buffer Object

If the template data is pulled from a template data file, the **LinkTemplate** object contains the path to the data file. If the template data is pulled from a data buffer (MPLS only), the **LinkTemplate** object contains the **DataBuffer** object. The **DataBuffer** can contain values for any variable defined in the template definition.

In the following example, the values for the variables **Source-IP**, **Dest-IP**, and **protocol**, are defined in the **DataBuffer** object.

```
<objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">DataBuffer</className>
            <properties xsi:type="ns1:CIMPropertyList"
                        soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">Source-IP</name>
                  <value xsi:type="xsd:string">132.235.123.0</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">Dest-IP</name>
                  <value xsi:type="xsd:string">54.103.63.0</value>
              </item>
             <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">protocol</name>
                  <value xsi:type="xsd:string">udp</value>
              </item>
```

You can also use the **DataBuffer** to specify values for variables defined elsewhere in the service request. Instead of entering the variable and value in the service request and then repeating them again in the **LinkTemplate**, you can simply call the value using the **DataBuffer**.

In the following partial example for an MPLS service request, in the **LinkAttrs** class, values are listed for **PE_VCI**, **PE_BGP_AS_ID**, and **Max_route_threshold**.

```
<objectPath xsi:type="ns1:CIMObjectPath">
               <className xsi:type="xsd:string">LinkAttrs</className>
                <properties xsi:type="ns1:CIMPropertyList"
                       soapenc:arrayType="ns1:CIMProperty[]">
                <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">CE_Intf_Name</name>
                     <value xsi:type="xsd:string">ATM1.22</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">PE_Intf_Name</name>
                    <value xsi:type="xsd:string">Switch1.234</value>
                </item>
               <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">PE_VCI</name>
                    <value xsi:type="xsd:string">234</value>
                </item>
               <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">PE_BGP_AS_ID</name>
                    <value xsi:type="xsd:string">13979</value>
                </item>
               <item xsi:type="ns1:CIMProperty">
                   <name xsi:type="xsd:string">Max_route_threshold</name>
                   <value xsi:type="xsd:string">25</value>
```

```
            </item>
          </properties>
        </objectPath>
```

The next section of this example shows these same values being called again in the **DataBuffer**.

```
<objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">DataBuffer</className>
            <properties xsi:type="ns1:CIMPropertyList"
                        soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">PE_VCI</name>
                  <value xsi:type="xsd:string">$PE_VCI</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">PE_BGP_AS_ID</name>
                  <value xsi:type="xsd:string">$PE_BGP_AS_ID</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">Max_route_threshold</name>
                  <value xsi:type="xsd:string">$Max_route_threshold</value>
              </item>
```

Note See the Repository Variable Chooser in the GUI Template Manager for a list of variables, by service blade, that can be recalled by the **DataBuffer**. (From the Service Design tab, click the Templates link. Choose the template Data file and click **Vars** on the Data File Editor window.)

In Table 4-7, the required parameters listed for **ServiceRequestDetails** are only for the template portion of the service order. For more information on service requests, see the appropriate chapters on service provisioning in this guide.

Table 4-7 Templates in a Service Request

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=<choose the appropriate service type> • ServiceRequestDetails
	ServiceRequestDetails	• MplsVpnLink (or **ACAttr**)

Table 4-7 Templates in a Service Request (continued)

Operation	className	Required Parameters
	MplsVpnLink (or the link object for your service)	• LinkTemplate
	LinkTemplate	• DatafilePath=<the pathname to the template definition folder> • LogicalDevice • The template data information, either from a data file or a data buffer. – DatafileName=<the pathname/filename to the template data file> – DataBuffer=<template data> (The name/value pairs.) • TemplateActive=true • TemplateAction= – APPEND – PREPEND

Note The attributes **PE_Template**, **PE_Intf_Template**, **CE_Template**, and **CE_Intf_Template** allow NBI access to variables designed to hold template blobs (template blobs were used during MPLS provisioning in legacy versions of ISC).

XML Examples:

- CreateMPLSTemplateServiceOrder.xml
- CreateL2VPNTemplateServiceOrder.xml

Removing Template Configurations

When you modify a service request that has templates, or before you can decommission a service request that has templates, you must first remove the template information from the service request. This is accomplished using negate templates to remove the template configuration from the device.

Modifying a Service Request with Templates

To modify a template in an existing service request, the following tasks must occur in the order listed:

1. Turn off templates. This action changes the **TemplateActive** attribute for the template from **true** to **false**.
   - For MPLS service requests, the **modifyInstance** subaction automatically toggles the **TemplateActive** attribute to **false**.
   - For all other service requests, the **TemplateActive** attribute must be specifically set to **false** in the **modifyInstance** subaction.
2. Add negate templates. This action removes the template information from the device. Use the **createInstance** subaction to add a negate template.
3. Add new templates. This action adds a new template to the service request. Use the **createInstance** action to add templates.

Note You should wait until the task has completed (you receive a task completed message) before you run the next task.

Turning off Templates (for MPLS Service Requests)

When you create the MPLS service request with a template, ISC sets the attribute **TemplateActive=true**. To turn off the template, the attribute needs to be changed to **TemplateActive=false**.

For templates in an MPLS service request, this is accomplished using a **modifyInstance** subaction. When you execute a **modifyInstance** subaction on a template, ISC automatically changes the status of the attribute to **TemplateActive=false**.

Note This automatic change in the template attribute only occurs when you use a **modifyInstance** on a template in an MPLS service request. For other service type, see the "Turning off Templates (for All Other Service Types)" section on page 4-14.

Include the device that received the template configuration and the template name (**DataFilePath**) from the service request where the template was implemented. See the following example:

```
<objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
               <className xsi:type="xsd:string">LinkTemplate</className>
               <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                      soapenc:arrayType="ns1:CIMKeyProperty[]">
                 <item xsi:type="ns1:CIMProperty">
                   <name xsi:type="xsd:string">LogicalDevice</name>
                   <value xsi:type="xsd:string">PE-POP1</value>
                 </item>
                 <item xsi:type="ns1:CIMProperty">
                   <name xsi:type="xsd:string">DatafilePath</name>
                   <value xsi:type="xsd:string">/Examples/templ1-enable</value>
                 </item>
```

In this XML example, the template name /Examples/temp11-enable, for device PE-POP1, is turned off (**TemplateActive=false**) by the **modifyInstance** subaction.

Turning off Templates (for All Other Service Types)

Unlike with MPLS, this attribute change does not happen automatically for all other service types. You must use a **modfyInstance** subaction and include the attribute **TemplateActive=false** in the XML request. See the following example:

```
<objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
               <className xsi:type="xsd:string">LinkTemplate</className>
               <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                      soapenc:arrayType="ns1:CIMKeyProperty[]">
                 <item xsi:type="ns1:CIMProperty">
                   <name xsi:type="xsd:string">LogicalDevice</name>
                   <value xsi:type="xsd:string">PE-POP1</value>
                 </item>
               <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">TemplateActive</name>
                    <value xsi:type="xsd:string">false</value>
                </item>
               <item xsi:type="ns1:CIMProperty">
                   <name xsi:type="xsd:string">DatafilePath</name>
                   <value xsi:type="xsd:string">/Examples/templ1-enable</value>
                 </item>
```

Adding Negate Templates

After the template is turned off (changed to **TemplateActive=false**), you add a negate template using a **createInstance** subaction to remove the template information from the device. The **DataFilePath** of the negate template must be different from the original template.

See the following example:

```
<objectPath subAction="createInstance" xsi:type="ns1:CIMObjectPath">
               <className xsi:type="xsd:string">LinkTemplate</className>
                <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                      soapenc:arrayType="ns1:CIMKeyProperty[]">
                </keyProperties>
                 <properties xsi:type="ns1:CIMPropertyList"
                       soapenc:arrayType="ns1:CIMProperty[]">
                <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">LogicalDevice</name>
                    <value xsi:type="xsd:string">PE-POP1</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">DatafilePath</name>
                    <value xsi:type="xsd:string">/Examples/tempnegate</value>
                </item>
```

Adding New Templates

When the original template information is disabled and removed from the device, you can add new template information using:

- A **createInstance** to create the service order to modify the service request.
- A **modifyInstance** to modify the service request and service request details.
- A **createInstance** subaction to add the new template.

Note You are not required to create a new service order to add new templates. In one modify service request, you can turn off templates, add negate templates, and add new templates. However, you must keep the correct order of operations (turn off, add negate, then add new).

See the following example:

```
<ns1:performBatchOperation>
    <actions xsi:type="ns1:CIMActionList"
             soapenc:arrayType="ns1:CIMAction[]">
     <action>
      <actionName xsi:type="xsd:string">createInstance</actionName>
      <objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">ServiceOrder</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">ServiceName</name>
         <value xsi:type="xsd:string">Acme-Template1</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">CarrierId</name>
         <value xsi:type="xsd:string">22</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">DesiredDueDate</name>
         <value xsi:type="xsd:dateTime">2002-12-13T14:55:38.885Z</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">Organization</name>
         <value xsi:type="xsd:dateTime">NbiCustomer</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">NumberOfRequests</name>
         <value xsi:type="xsd:string">1</value>
        </item>
       </properties>
      </objectPath>
     </action>
     <action>
      <actionName xsi:type="xsd:string">modifyInstance</actionName>
      <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">ServiceRequest</className>
      <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
       <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">RequestName</name>
        <value xsi:type="xsd:string">Template1</value>
       </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">Type</name>
         <value xsi:type="xsd:string">Mpls</value>
        </item>
       </properties>
       <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
           <className xsi:type="xsd:string">ServiceRequestDetails</className>
            <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                      soapenc:arrayType="ns1:CIMKeyProperty[]">
                  <item xsi:type="ns1:CIMProperty">
                    <name xsi:type="xsd:string">LocatorId</name>
                    <value xsi:type="xsd:string">36</value>
                  </item>
```

```
 </keyProperties>
 <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">MplsVpnLink</className>
   <keyProperties xsi:type="ns1:CIMKeyPropertyList"
         soapenc:arrayType="ns1:CIMKeyProperty[]">
      <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">LocatorId</name>
        <value xsi:type="xsd:string">33</value>
      </item>
   </keyProperties>
   <properties xsi:type="ns1:CIMPropertyList"
          soapenc:arrayType="ns1:CIMProperty[]">
   </properties>
   <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">LinkAttrs</className>
   <keyProperties xsi:type="ns1:CIMKeyPropertyList"
         soapenc:arrayType="ns1:CIMKeyProperty[]">
  </keyProperties>
  <properties xsi:type="ns1:CIMPropertyList"
          soapenc:arrayType="ns1:CIMProperty[]">
  </properties>
 </objectPath>
<objectPath subAction="createInstance" xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">LinkTemplate</className>
  <keyProperties xsi:type="ns1:CIMKeyPropertyList"
         soapenc:arrayType="ns1:CIMKeyProperty[]">
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">LogicalDevice</name>
      <value xsi:type="xsd:string">PE-POP1</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">DatafilePath</name>
      <value xsi:type="xsd:string">/Examples/templ4-enable</value>
    </item>
  </keyProperties>
  <properties/>
  <!-- objectPath xsi:type="ns1:CIMObjectPath">
<className xsi:type="xsd:string">DataBuffer</className>
<properties xsi:type="ns1:CIMPropertyList"
          soapenc:arrayType="ns1:CIMProperty[]">
  <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">DLCI</name>
      <value xsi:type="xsd:string">20</value>
  </item>
```

Decommissioning a Service Request

When you decommission a service request with templates, you must first remove the template information from the service request.

The following processes must occur, in the order listed, to decommission a service request:

- Modify the service request to turn off (**TemplateActive=false**) the templates. Refer to the appropriate section for information:
  - "Turning off Templates (for MPLS Service Requests)" section on page 4-13
  - "Turning off Templates (for All Other Service Types)" section on page 4-14
- Create a service order to decommission the service request. Refer to the "Service Decommission" section on page 3-10 for more information.

Monitoring APIs

The IP Solution Center (ISC) provides methods for monitoring services. Monitoring APIs include event notifications, canned reports and SQL queries, SLA monitoring, and task logs. Monitoring APIs provide real time information for service providers.

This chapter contains the following sections:

Event Notifications

The API uses a notification server to deliver database change events. An event is registered and a notification is sent any time a database object is created, modified, or deleted, when a scheduled task begins or ends its execution, or when a watchdog event signals a change in execution status for any ISC server.

The event types are:

- InstCreation— An object has been created.
- InstDeletion—An object has been deleted.
- InstModification—An object has been modified.
- InstStateChange—A change in execution status for any ISC server.

The notification server listens to the Tibco bus for interested database change events and sends a notification to the client. When an event is recognized, the daemon processes the event and generates the appropriate indication XML response. The XML response is delivered either back across the client connection or to a specified URL.

Setting up the Client

Note An example client, EventListener, provided with the ISC software, is a simple servlet that listens to the notifications servlet. Use this example to create your own client for event notifications. See Appendix B, "Implementing a Notification Server," for more information.

The client must register for events. Use the following properties to enable notifications and to indicate where notification messages should be sent.

- notification.clientEnabled—Used to turn notification forwarding on and off. To enable the client, set **notification.clientEnabled=true**.
- notification.clientHost—The machine running the event notification receiving program.
- notification.clientPort—The listener port to open on the receiving machine.
- notification.clientMethod—The method, or program, to contact on the receiving machine. The clientMethod is defined in the Tomcat web.xml file. The notification web.xml file (located in $ISC_HOME/resources/webserver/tomcat/webapps/notification/WEB-INF) identifies two servlets. One is the notificationServlet and the other is the eventListener (EventReceivingServlet program) servlet.
  - The notificationServlet is the ISC program responsible for collecting and forwarding events of interest.
  - The clientHost, clientMethod, and clientPort are used to construct a URL.
  - The clientMethod/notification/servlet/eventListener is mapped to the eventListener servlet.
- notification.clientRegFile—Located in /opt/vpnsc/iscadmin/resources/nbi/notification/clientReg.txt, this file contains the events of interest to be forwarded. See Appendix B, "Implementing a Notification Server," for a complete list of the events that can be collected.

  To define the events are of interest to the client, change the **notification.clientRegFile** so that it points to a file that contains all interested events. For example, if you have the following lines in this file:

  ```
  com.cisco.vpnsc.repository.devices.PIX.add
  com.cisco.vpnsc.repository.devices.PIX.modify
  com.cisco.vpnsc.repository.task.PersistentTask.>
  ```

  The first line defines events that add PIX objects. The second line defines events that modify PIX objects. The third line defines all **PersistentTask** related events. (The ">" symbol is a wildcard to represent any match.)

Tip Use **com.cisco.vpnsc.repository.>** to represent any repository change event.

Remote Authentication

The notification server allows ISC events to be delivered using an HTTP interface to a remote system. If your remote system requires authentication, use these properties to set up the remote user and password information.

- notification.remoteUsername

- notification.remotePassword

These properties allows the ISC notification server to respond to remote authentication requests, which is required by certain systems prior to establishing a connection. The default values for these properties is null.

Notification Responses

Event notifications are sent in the form of XML responses. The operation for event notification is **deliverEvent**. For each event, the following information is returned in the XML response:

- IndicationTime
- IndicationType=
  - InstCreation
  - InstDeletion
  - InstModification
  - InstStateChange (for service requests)
- InstClassName
- LocatorId
- Name

If the state of a service request changes, additional information is returned under **InstIndicationDetails**:

- Previous_SR_State
- Current_SR_State
- Description

See the following example:

```
<objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">InstIndicationDetails</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item>
         <name xsi:type="xsd:string">Previous_SR_State</name>
         <value xsi:type="xsd:string">PENDING</value>
        </item>
        <item>
         <name xsi:type="xsd:string">Current_SR_State</name>
         <value xsi:type="xsd:string">ACTIVE</value>
        </item>
        <item>
         <name xsi:type="xsd:string">Description</name>
         <value xsi:type="xsd:string">System is active</value>
        </item>
       </properties>
        </objectPath>
       </objectPath>
```

Reports

The ISC API includes a reporting feature for queries on inventory, topology, and services. Reports can be generated from SQL-based queries or from canned reports that are provided with the ISC product.

Note ISC's reporting feature supports Sybase and Oracle databases.

The ISC API can generate reports for:

- Inventory
  - Physical inventory
  - VLAN usage reports
  - Customer reports
  - Site reports
  - Resource usage reports
- Topology reports:
  - Device connectivity
  - L2VPN topology
  - MPLS topology
  - VPLS topology
- Service reports
  - Service request reports
  - Service order reports

Generating reports from meta-based SQL queries (**execQuery**) and from canned reports (**execReport**) are described in the following sections. SLA Reports are described in "SLA Reports" section on page 5-18.

SQL-Based Reports

The **execQuery** operation allows you to execute a direct search of the ISC database and to specify the form of the output data. The query language is a subset of SQL and supports queries specified with parameters and objects defined in the XML schema. A response to an **execQuery** returns data records and can be sent in an XML response or saved to a file. File data can be in either XML format or comma separated value (CSV) format. See "Output for Reports" section on page 5-10 for more information.

Note You can specify a delimiter other than commas, if necessary. The ISC API supports a single character delimiter.

Use the **execQuery** operation to query the main ISC repository or the SLA repository.

- The main ISC repository holds data for objects representing devices and VPN network elements. It also contains data collected from devices by the collection server, and task data, which is used by the scheduler.
- The SLA repository contains SLA tables populated with the data gathered by SLA probes.

The SQL search criteria for **execQuery** is defined as follows:

- *Select* property list—A comma-separated list of property names related to the individual classes specified in the *From* clause. An asterisk (*) can be used to specify ALL of the properties of a class.
- *From* class list—A comma-separated list of class names. If more than one class name is specified, the classes must be related by an association, which is a condition in the *Where* clause.
- *Where* conditions—Specifies the criteria by which results are selected. The criteria can be simple property comparisons.
- *Orderby* clause—Specifies the criteria by which results are sorted.

Tip If you are using an Oracle database, you should use aliases for your *Select* names because it has a 30 character limit.

Note The **execQuery** operation does not support aggregate functions or subqueries. Use the *where* clause for joins.

The body of the XML request contains the **execQuery** operation, the **QueryLanguage**, and the SQL search criteria, specified with the **Query** attribute. See the following example:

```
<ns1:execQuery>
      <objectPath xsi:type="ns1:CIMObjectPath">
        <className xsi:type="xsd:string">execQuery</className>
        <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                  soapenc:arrayType="ns1:CIMKeyProperty[]">
          <item xsi:type="ns1:CIMKeyProperty">
            <name xsi:type="xsd:string">QueryLanguage</name>
            <value xsi:type="xsd:string">WQL</value>
          </item>
          <item xsi:type="ns1:CIMKeyProperty">
            <name xsi:type="xsd:string">Query</name>
            <value xsi:type="xsd:string">select Id, Name, ContactInfo from
Organization</value>
          </item>
        </keyProperties>
      </objectPath>
    </ns1:execQuery>
```

In this example:

- The operation is **execQuery**
- In the **execQuery** class, the **QueryLanguage** is **WQL**
- The search criteria for the **execQuery** class is; *Select* properties, **Id**, **Name**, and **ContactInfo** *From* any record with the class name **Organization**.

An **execQuery** request with the above search criteria returns the following records in XML format:

```
<record>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">Record#1</className>
            <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Organization.Name</name>
                <value xsi:type="xsd:string">NbiCustomer</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
```

```
                <name xsi:type="xsd:string">Organization.Id</name>
                <value xsi:type="xsd:string">1</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Organization.ContactInfo</name>
                <value xsi:type="xsd:string">Mrs Brown, Boulder, Colorado</value>
              </item>
            </properties>
          </objectPath>
        </record>
        <record>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">Record#2</className>
            <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Organization.Name</name>
                <value xsi:type="xsd:string">cust_for_greymgmt_NbiProvider_0</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Organization.Id</name>
                <value xsi:type="xsd:string">2</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Organization.ContactInfo</name>
                <value xsi:type="xsd:string">Cust for GreyMgmtVpn for
Provider=NbiProvider</value>
              </item>
            </properties>
          </objectPath>
        </record>
```

Canned Reports

ISC provides canned reports for frequently requested report data. These canned reports can be used as is, copied, modified, or extended by customers. Canned reports are located in the $ISC_HOME/resources/nbi/reports/ISC directory. This is the defaul location for canned reports.

Custom reports, which are canned reports that have been modified for a particular customer, are located in the $ISC_HOME/resources/nbi/reports/custom directory. To use custom reports, you must change the property file to so that it points to this directory. See the following example:

```
nbi.CustomerReportMetaDir=/opt/isc/resources/nbi/reports/custom
```

Table 5-1 describes the types of canned reports provided with ISC.

Table 5-1 Canned Reports Provided with ISC

Report Type	Response Data
Service path information	Customer ID, service order number, locator ID, circuit path per access leg, including the device ID, VLAN ID, Port ID, VC ID for the PE-CLE, PE-POP, and all intermediate devices.
Service class and service path information	Customer ID, service order number, locator ID, service level or service class, committed information rate (CIR), peak information rate (PIR), and the service path information.

Table 5-1 Canned Reports Provided with ISC (continued)

Report Type	Response Data
Detailed service configuration information	Customer ID, service order number, locator ID, port ID, assigned VLAN ID, CIR and PIR for the PE-CLEs and PE-POPs that are the UNI endpoints.
MPLS VPN association information	Interface name, IP address, and index, VLAN and VC IDs, and physical interface name for the PE-POPs, CE ID, Customer ID, VRF and VPN information, and service request state and locator ID.
Inventory queries	For the entire network, by access domain, or by network element.
List of assigned VLAN Ids	UNI (port ID) and VLAN IDs.
List of available VLAN Ids	VLAN IDs.
List of available PE-CLEs within an access domain	PE-POP and PE-CLE IDs.
List of available UNIs within an access domain	UNI and network element ID.
List of PE-CLEs within an access domain	PE-POP and PE-CLE IDs.
List of assigned UNIs	Customer ID, network element ID, and UNI.
List of assigned services	Customer ID and service type.
Service order status	Customer ID, service order number, locator ID, status and due date.

Canned reports use the **execReport** operation in the XML request. The XML request provides the search criteria, and the response returns the report data.

See the following example of an XML request for the canned report; **VPNReport**:

```
</soapenv:Header>
  <soapenv:Body>
   <ns1:execReport >
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">VPNReport</className>
     <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                  soapenc:arrayType="ns1:CIMKeyProperty[]">
       <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">customer.name</name>
        <value xsi:type="xsd:string">Nbi%</value>
       </item>
       <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">vpn.name</name>
        <value xsi:type="xsd:string">vpnX</value>
       </item>
       <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">pe_cisco_router.host_name</name>
        <value xsi:type="xsd:string">enswosr1</value>
       </item>
       <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">mpls_sr.sr_state</name>
        <value xsi:type="xsd:string">FAILED_DEPLOY</value>
       </item>
     </keyProperties>
     <sortList xsi:type="ns1:CIMPropertyList"
             soapenc:arrayType="ns1:CIMProperty[]">
       <itemName xsi:type="xsd:string">pe_cisco_router.host_name:asc</itemName>
```

```
        <itemName xsi:type="xsd:string">vpn.name:desc</itemName>
        <itemName xsi:type="xsd:string">dev_endpoint.intf_name</itemName>
      </sortList>
     </objectPath>
    </ns1:execReport>
   </soapenv:Body>
</soapenv:Envelope>
```

Report Definitions

The search criteria for a canned report is derived from a report definition. The search criteria in the example XML request, **vpn.name**, **pe_cisco_router.host_name**, and **mpls_sr.sr_state,** correlate to parameters in the the report definition (meta file). Each canned report has an associated report definition file.

The report definition specifies report header information, search criteria, parameter definitions, filters, and sort order for the report output. The following sections show the different parts in a report definition.

Header

The header in the report definition lists the report name, the label, and report title.

```
<packageDef name="MPLS">
<objectDef name="VPNReport"
           label="VPNReport"
           title="MPLS VPN Association report"
           inSchema="report"
           securityOn="true"
           sql="
```

Note There is no RBAC record filter in the current version of ISC.

Search Criteria

The search criteria includes:

- *Select* property list—A comma-separated list of property names related to the individual classes specified in the *From* clause. An asterisk (*) can be used to specify ALL of the properties of a class.
- *From* class list—A comma-separated list of class names. If more than one class name is specified, the classes must be related by an association, which is a condition in the *Where* clause.
- *Where* conditions—Specifies the criteria by which results are selected. The criteria can be simple property comparisons.
- The search criteria in the report definition is the same as for execQuery

```
SELECT distinct
pe_cisco_router.host_name,
pe_endpoint.ip_address,
customer.name,
mpls_sr.id,
mpls_sr.sr_state,
vpn.name,

FROM
customer,
cerc JOIN  vpn
    ON cerc.vpn_id = vpn.id,
```

```
vrf_role JOIN  cisco_router as pe_cisco_router
    ON vrf_role.device_id = pe_cisco_router.id,
mpls_vpn_link JOIN  mpls_sr
    ON mpls_vpn_link.sr_id = mpls_sr.id,
mpls_vpn_link LEFT OUTER JOIN endpoint AS pe_endpoint
    ON mpls_vpn_link.pe_ep_id = pe_endpoint.id
WHERE mpls_sr.subsumed_by IS NULL
AND pe_cisco_router.id = pe_dev_ifc.device_id
AND vpn.customer_id = customer.id {and (filterSet1 or filterSet2)}">
```

Note The filter sets in this meta definition (*filterSet1* and *filterSet2*) are defined with the filter parameters (see Filters), and referenced in the SQL search criteria. See Named Filter Sets, page 5-10 for more information.

Parameter Definitions

The parameter definitions associate objects and definitions in the ISC meta file to use in the canned report. Parameter definitions can also define access parameters for each object and filter options. The parameters definitions used in the example XML request are indicated in bold.

```
<paramDef name="pe_cisco_router.host_name"
              objectRefName="CiscoRouter"
              objectRefParamName="HostName"
              access="create_na, modify_na, view_ro"
              label="PE Device Name" />

    <paramDef name="customer.name"
              objectRefName="Customer"
              objectRefParamName="Name"
              mandatory="true"
              filterOperator="like"
              access="create_na, modify_na, view_ro"
              label="Customer Name"/>

    <paramDef name="mpls_sr.id"
              objectRefName="MplsSR"
              objectRefParamName="JobId"
              access="create_na, modify_na, view_ro"
              label="SR Locator Id" />

    <paramDef name="mpls_sr.sr_state"
              objectRefName="MplsSR"
              objectRefParamName="State"
              access="create_na, modify_na, view_ro"
              label="SR State"/>

    <paramDef name="vpn.name"
              objectRefName="VPN"
              label="VPN Name"
              access="create_na, modify_na, view_ro" />
```

Filters

The filter section lists the parameters that filters can to be applied to in this report definition. The items in bold indicate the parameters used in the example XML request.

```
<paramSetDef name="filter">
    <paramSetItem name="pe_cisco_router.host_name" />
    <paramSetItem name="customer.name" />
    <paramSetItem name="mpls_sr.id" />
```

```
    <paramSetItem name="vpn.name" />
    <paramSetItem name="mpls_sr.sr_state" />
</paramSetDef>
```

Named Filter Sets

There can be additional filters with special references in the meta definition, called named filter sets. Named filter sets can be used when an SQL subqueries has its own *where* clause that requires user input. You can apply operators, arguments, and attributes (such as *mandatory*), to filters sets in the meta definition.

```
<paramSetDef name="filterSet1">
<paramSetItem name="pe_cisco_router.host_name" />
</paramSetDef>

<paramSetDef name="filterSet2">
<paramSetItem name="ce_cisco_router.host_name" />
</paramSetDef>
```

Sorting

This section of the report definition defines the default sort criteria. Use the paramDef *name* (not the label) to identify the parameters to sort. Note that the PE Device Name is sorted in ascending order (asc), and VPN Name is sorted in descending order (decs).

```
<paramSetDef name="sort">
    <paramSetItem name="pe_cisco_router.host_name:asc" />
    <paramSetItem name="vpn.name:desc" />
</paramSetDef>
</objectDef>
</packageDef>
```

Output for Reports

The response to an **execQuery** or **execReport** request can be sent as an XML response or exported to an output file on the ISC server. File data can be in XML format or comma separated value (CSV) format.

- Output in XML format is a typical XML response that contains data records separated by record elements. The XML format is the default for **execQuery** or **execReport** output data.
- Output in CSV format is data records separated by commas (or another specified delimiter). The first line of the output is the list of column labels. The following lines contain the records. See the following example:

  ```
  Organization.Name, Organization.Id, Organization.ContactInfo
  NbiCustomer, 1, Mrs Brown Boulder Colorado
  cust_for_greymgmt_NbiProvider_0, 2, Cust for GreyMgmtVpn for Provider=NbiProvider
  ```

To specify CSV format for the output data, you must add the filename, format, and delimiter in the first line of the XML response. To have the output data sent to a file (XML or CSV data), you define the filename and format in the first line of the **execQuery** and **execReport** XML requests. The first line is processed during the output and not during the query operation. See the following example:

```
<ns1:execQuery filename="/users/user1/tmp/querydata.xml" format="csv" delimiter=";"
maxRecords = "100">
      <objectPath xsi:type="ns1:CIMObjectPath">
        <className xsi:type="xsd:string">execQuery</className>
        <keyProperties xsi:type="ns1:CIMKeyPropertyList"
```

```
                soapenc:arrayType="ns1:CIMKeyProperty[]">
          <item xsi:type="ns1:CIMKeyProperty">
            <name xsi:type="xsd:string">QueryLanguage</name>
            <value xsi:type="xsd:string">WQL</value>
          </item>
          <item xsi:type="ns1:CIMKeyProperty">
            <name xsi:type="xsd:string">Query</name>
            <value xsi:type="xsd:string">select a.Name, b.* from Organization a, Site b
where a.Id=b.Customer_Id order by a.Name</value>
          </item>
        </keyProperties>
      </objectPath>
    </ns1:execQuery>
```

In this example:

- **filename=”/users/user1/tmp/querydata”** is the location of the output file name.
- **format=”csv”** specifies that the output be in CSV format
- **delimiters=”;”** specifies that the output data be separated by semi-colons. This parameter is optional.
- **maxRecords=”100”** specifies that the API return no more than 100 records. This parameter is optional.

Use the following guidelines when choosing the format of the output data:

- If you specify XML, or do not specify a format, the output is returned in the form of an XML response.
- If you specify CSV, and send the output to a file, the output is CSV data only, no XML tags. This data cannot be sent across an HTTP connection. CSV data can be imported into a spreadsheet, or any reporting tool that supports a one character delimiter.
- If the file already exists, the response data overwrites this existing file.
- If you specify an output file, the XML response sent across the client connection lists the name of the file and a line count of the output data.
- If you specify an output file, report data is written to a file on the ISC server.

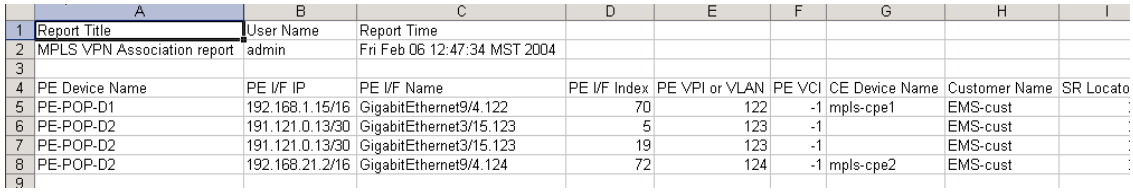
Note The line count equals the number of records, plus one (the line for the column labels).

- If the API cannot write to this file, you receive an error.
- If the maximum record limit is reached, the data is truncated at the record limit. You receive the following message, which includes the maximum records value that has been exceeded.

```
<error xsi:type="ns1:CIMError">
              <code xsi:type="xsd:int">1001</code>
              <description xsi:type="xsd:string">Max records exceeded</description>
              <detail xsi:type="xsd:string"> Max number of records = 4096</detail>
            </error>
```

The report headers in the output for a canned report, include report title, creation time, and the user associated with the session. Figure 5-1 shows the output for the MPLS VPN Association Report sent to a CSV file and displayed in a spreadsheet.

Figure 5-1 Example of an MPLS VPN Association Report



	A	B	C	D	E	F	G	H	I
1	Report Title	User Name	Report Time						
2	MPLS VPN Association report	admin	Fri Feb 06 12:47:34 MST 2004						
3									
4	PE Device Name	PE I/F IP	PE I/F Name	PE I/F Index	PE VPI or VLAN	PE VCI	CE Device Name	Customer Name	SR Locato
5	PE-POP-D1	192.168.1.15/16	GigabitEthernet9/4.122	70	122	-1	mpls-cpe1	EMS-cust	
6	PE-POP-D2	191.121.0.13/30	GigabitEthernet3/15.123	5	123	-1		EMS-cust	
7	PE-POP-D2	191.121.0.13/30	GigabitEthernet3/15.123	19	123	-1		EMS-cust	
8	PE-POP-D2	192.168.21.2/16	GigabitEthernet9/4.124	72	124	-1	mpls-cpe2	EMS-cust	
9									

114379

SLA Provisioning

A service level agreement (SLA) defines the level of service provided to a customer by a service provider. ISC can monitor the service-related performance criteria by provisioning, collecting, and monitoring SLAs on Cisco IOS routers that support the Service Assurance Agent (SA Agent) devices.

ISC uses the Cisco SA Agent MIB to monitor SLA metrics. The SA Agent allows you to configure probes for performance measurements and uses a router monitor (RTRMON) MIB for access through simple network management protocol (SNMP). The MIB also supports SNMP notifications.

The SLA server monitors network performance by measuring response time, jitter, connect time, throughput, and packet loss. The API supports configuring SLA probes and creating SLA reports to retrieve the data collected by the probes.

Note To collect SLA data, a device must have SA Agent and SNMP enabled, and have SNMP parameters set. To use the jitter (voice jitter) protocol to collect SLA data from the edge devices in your network, enable SA Agent on each device from which you want to collect this data.

Note You must have IP connectivity between SA Agent devices.

Process for SLA Provisioning

The following process summarizes the SLA provisioning process using the API.

1. Create and deploy the SLA probes. See the "Creating and Deploying SLA Probes" section on page 5-13.
2. Modify the SLA probe to enable or disable traps. Use ModifySLAProbe.xml.
3. Run an SLA collection task to start the collection of SLA data. See the "SLA Collection" section on page 3-12.
4. Gather the SLA data using the processes for SQL-Based Reports **or** Canned Reports. ISC collects the data from the appropriate SLA databases. See the "SLA Reports" section on page 5-18.
5. ISC returns the data across the client HTTP connection in an XML response or saves it to an output file. The output file can be in XML or CSV format. See the "Output for Reports" section on page 5-10.
6. This process is optional. Set up the API to receive event notifications for SLA probe and SLA collection changes (add/delete/modify instances). See the "Event Notifications" section on page 5-1.

SLA Probes

SLA probes can be configured on Cisco IOS devices. The probes store the measurement parameters at a certain frequency (for example, every 5 minutes) in the IOS buffer. An SLA collection server retrieves the measurement data from the device buffer at hour intervals and stores it in the ISC database. There is one database for each collection server.

Using the ISC API, you can create, delete, or view an SLA probe. You can modify a probe, but only to change the value of the **EnableProbe** or **EnableTraps** attributes (true/false).

ISC supports creating probes from the following devices:

- From any SA Agent device
- From MPLS CPE
- From MPLS PE or MVRF-CE

To retrieve the measurement data from the database, use an SLA report. See the "SLA Reports" section on page 5-18 for more information.

Creating and Deploying SLA Probes

To create an SLA probe from any SA Agent device, specify the common probe parameters, source devices, destination devices (where applicable), and protocol (**ProbeType**).

To create an SLA probe from an MPLS CPE, PE, or MVRF-CE, you must also specify a VPN. For MPLS PEs and MPLS MVRF-CEs, a VRF is also required.

Table 5-2 shows the operation, classNames, and child attributes to specify to create and deploy an SLA probe.

Table 5-2 Create SLA Probe

Operation	className	Children
performBatchOperation		
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=SLAProbe • ServiceRequestDetails
	ServiceRequestDetails	• Common Probe Parameters • SourceDevice • DestinationDevice (where applicable) • VPN (only for MPLS PE, MPLS CPE, or MPLS MVRF-CE) • VRF (only for MPLS PE or MPLS MVRF-CE) • Probe Types (protocol)

See the following example:

```
<soapenv:Body>
    <ns1:performBatchOperation>
      <actions xsi:type="ns1:CIMActionList"
              soapenc:arrayType="ns1:CIMAction[]">
        <action>
          <actionName xsi:type="xsd:string">createInstance</actionName>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">ServiceOrder</className>
            <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">ServiceName</name>
                <value xsi:type="xsd:string">SLAProbeSR</value>
              </item>
             <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">DesiredDueDate</name>
                <value xsi:type="xsd:dateTime">2003-04-10T14:55:38.885Z</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">NumberOfRequests</name>
                <value xsi:type="xsd:dateTime">1</value>
              </item>
             </properties>
          </objectPath>
        </action>
        <action>
          <actionName xsi:type="xsd:string">createInstance</actionName>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">ServiceRequest</className>
            <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">RequestName</name>
                <value xsi:type="xsd:string">SLAProbeSR</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Type</name>
                <value xsi:type="xsd:string">SLAProbe</value>
              </item>
            </properties>
            <objectPath xsi:type="ns1:CIMObjectPath">
              <className xsi:type="xsd:string">ServiceRequestDetails</className>
              <properties xsi:type="ns1:CIMPropertyList"
                        soapenc:arrayType="ns1:CIMProperty[]">
                 <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">ProbeThreshold</name>
                     <value xsi:type="xsd:string">5000</value>
                 </item>
                 <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">ProbeTimeout</name>
                     <value xsi:type="xsd:string">5000</value>
                 </item>
                 <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">ProbeLife</name>
                     <value xsi:type="xsd:string">-1</value>
                 </item>
                 <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">ProbeFrequency</name>
                     <value xsi:type="xsd:string">5500</value>
                 </item>
                 <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">ProbeTOSType</name>
```

```
                        <value xsi:type="xsd:string">PRECEDENCE</value>
                      </item>
                      <item xsi:type="ns1:CIMProperty">
                        <name xsi:type="xsd:string">ProbeTOSValue</name>
                        <value xsi:type="xsd:string">0</value>
                      </item>
                      <item xsi:type="ns1:CIMProperty">
                        <name xsi:type="xsd:string">ProbeKeepHistoryFlag</name>
                        <value xsi:type="xsd:string">false</value>
                      </item>
                      <item xsi:type="ns1:CIMProperty">
                        <name xsi:type="xsd:string">EnableTraps</name>
                        <value xsi:type="xsd:string">false</value>
                      </item>
                    </properties>
                    <objectPath xsi:type="ns1:CIMObjectPath">
                      <className xsi:type="xsd:string">SourceDevice</className>
                      <properties xsi:type="ns1:CIMPropertyList"
                            soapenc:arrayType="ns1:CIMProperty[]">
                        <item xsi:type="ns1:CIMProperty">
                          <name xsi:type="xsd:string">SrcDevice</name>
                          <value xsi:type="xsd:string">slaDummyOne</value>
                        </item>
                        <item xsi:type="ns1:CIMProperty">
                          <name xsi:type="xsd:string">Interface</name>
                          <value xsi:type="xsd:string">Ethernet0/0</value>
                        </item>
                      </properties>
                    </objectPath>
                    <objectPath xsi:type="ns1:CIMObjectPath">
                      <className xsi:type="xsd:string">DestinationDevice</className>
                      <properties xsi:type="ns1:CIMPropertyList"
                            soapenc:arrayType="ns1:CIMProperty[]">
                        <item xsi:type="ns1:CIMProperty">
                          <name xsi:type="xsd:string">DestDevice</name>
                          <value xsi:type="xsd:string">slaDummyTwo</value>
                        </item>
                        <item xsi:type="ns1:CIMProperty">
                          <name xsi:type="xsd:string">Interface</name>
                          <value xsi:type="xsd:string">Ethernet1/1</value>
                        </item>
                      </properties>
                    </objectPath>
                    <objectPath xsi:type="ns1:CIMObjectPath">
                      <className xsi:type="xsd:string">EchoProbe</className>
                      <properties xsi:type="ns1:CIMPropertyList"
                            soapenc:arrayType="ns1:CIMProperty[]">
                        <item xsi:type="ns1:CIMProperty">
                          <name xsi:type="xsd:string">RequestDataSize</name>
                          <value xsi:type="xsd:string">32</value>
                        </item>
                      </properties>
                    </objectPath>
                  </objectPath>
                </objectPath>
              </action>
            </actions>
          </ns1:performBatchOperation>
        </soapenv:Body>
```

Common Probe Parameters

Table 5-3 describes the common probe parameters.

Table 5-3 Common Probe Parameters

Parameter	Description
Required Parameters	
ProbeLife	The number of seconds the probe is active. A value of **-1** indicates that the probe is active indefinitely.
ProbeThreshold	Threshold limit, in milliseconds. When this value is exceeded and traps are enabled, a trap is sent. If the SA Agent operation time exceeds this limit, a violation is recorded by the SA Agent. This value must not exceed the **ProbeTimeout** value.
ProbeTimeout	The time, in milliseconds, to wait for an SA Agent operation to complete. The value must be less than or equal to the **ProbeFrequency** value and greater than or equal to the **ProbeThreshold** value.
ProbeFrequency	The duration, in seconds, between initiating each SA Agent operation. The value must be greater than or equal to the **ProbeTimeout** value. The range is 0 to 604800.
ProbeTOSType {**PRECEDENCE** | **DSCP**}	The range is: • **PRECEDENCE**— 0 to 7 • **DSCP**—0 to 63.
ProbeTOSValue	The three most significant bits of the TOS field in an IP header.
Optional Parameters	
ProbeKeepHistoryFlag {**true** | **false**}	If true, ISC keeps the recent history table on the router. This is kept in the SA Agent MIB that keeps the raw round-trip time (RTT) SLA measurement. You must also specify the **ProbeHistoryBuckets**. **Note** Not supported for HTTP and Jitter reports.
ProbeHistoryBuckets	Required if **ProbeKeepHistoryFlag=true**. Indicates the number of most recent raw data entries to be kept in the raw history data. When the raw data entries value is exceeded, the oldest bucket is removed to keep the entries within the limit. The range is 1 to 60.
EnableTraps {**true** | **false**}	If true, the SLA is configured to send three types of traps. You must also specify **ProbeFallingThreshold**.
ProbeFallingThreshold	Required if **EnableTraps=true**. When traps are enabled and the delay meets this value, a trap is sent. The range is 1 to the **ProbeThreshold** value, in milliseconds.

Probe Types

ISC uses the **ProbeType** to specify the protocol of the traffic to monitor. The following protocols are available when you create an SLA probe from all devices *only* if destination devices are available:

- Internet Control Message Protocol Echo (ICMP Echo)
- Transmission Control Protocol Connect (TCP Connect)

- User Datagram Protocol Echo (UDP Echo)
- Jitter (voice jitter)

The following protocols are available when you create an SLA probe from all devices *except* MPLS PE:

- TCP Connect
- File Transfer Protocol (FTP)
- Domain Name System (DNS)
- Hyper text Transfer Protocol (HTTP)
- Dynamic Host Configuration Protocol (DHCP)

Table 5-4 describes the required attributes for each probe type (protocol).

Table 5-4 Attributes for Probe Types

Probe Type	Attributes
EchoProbe	• RequestDataSize
UDPEchoProbe	• RequestDataSize • DestinationPortNumber
DNSProbe	• NameServerAddress • NameToBeResolved • RequestDataSize
FTPProbe	• UserName • Password • HostIPAddress • FilePath
DHCPProbe	• No specific attributes
HTTPProbe	• HTTPVersion • HTTPUrl • EnableHTTPCacheFlag • HTTPProxyServer • NameServerAddress • HTTPOperation – HTTPGet – HTTPRaw (**HTTPRawRequest** is also required) • RequestDataSize
JitterProbe	• RequestDataSize • DestinationPortNumber • PacketCount • Interval
TCPConnectProbe	• RequestDataSize • DestinationPortNumber

Viewing SLA Probes

To view an SLA probe, use **enumerateInstances**, specify the probe type (for example, **EchoProbe**, **JitterProbe**, **HTTPProbe**), and enter a unique identifier, such as **LocatorID**.

Note The **LocatorID** can also be used to retrieve, modify, and delete a specific probe.

To view probes for a specific device, use **SrcDevice** or **DestDevice** as the unique identifier. See the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns0="http://www.cisco.com/cim-cx/2.0"
  xmlns:ns1="urn:CIM">
  <soapenv:Header>
   <ns0:message id="87855" timestamp="2002-12-13T14:55:38.885Z"
                sessiontoken="p36bttjwy1"/>
  </soapenv:Header>
  <soapenv:Body>
   <ns1:enumerateInstances>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">EchoProbe</className>
     <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                    soapenc:arrayType="ns1:CIMKeyProperty[]">
       <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">SrcDevice</name>
        <value xsi:type="xsd:string">slaDummyOne</value>
       </item>
     </keyProperties>
    </objectPath>
   </ns1:enumerateInstances>
 </soapenv:Body>
</soapenv:Envelope>
```

SLA Reports

SLA reports gather information from the various SLA tables in the SLA database. SLA data is derived at specific intervals from the SLA repository. The information gathered in SLA reports depends on the probes that have been set for collection of information.

SLA reports can be created using **execQuery** or **execReport.**

- **execQuery** retrieves raw data from the main ISC database or the SLA database. See the "SQL-Based Reports" section on page 5-4 for more information.
- **execReport** for SLA, generates SLA reports based solely on data from the SLA tables in the SLA database.

ISC supports the following report types:

- **SLASummaryReport**
- **SLAHTTPReport**
- **SLAJitterReport**

- **SLASummaryCoSReport**
- **SLAHTTPCoSReport**
- **SLAJitterCoSReport**

For the Summary, HTTP, and Jitter reports, you filter the SLA probe by DSCP and or IP precedence value. For the Summary CoS, HTTP CoS, and Jitter CoS reports, you specify the TOS type for the whole report.

To define the conditions for the SLA report, use the following required attributes:

- **ValueDisplayed**—See Table 5-5 for a complete list of options for each report type.
- **Timeline** and **TimeGranularity—All, Yearly, Monthly, or Daily, Hourly.**
- **AggregateBy—All**, **Customer**, **Provider**, **VPN**, **Source_Router**, or **Probe**.

The following filters are also available for SLA reports:

  - Organization
  - Provider
  - VPN
  - SrcDevice
  - DestDevice
  - Probes
  - TOS
  - DSCP

See the following example:

```
<soapenv:Body>
   <ns1:execReport>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">SLAJitterReport</className>
     <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                    soapenc:arrayType="ns1:CIMKeyProperty[]">
      <item xsi:type="ns1:CIMKeyProperty">
       <name xsi:type="xsd:string">ValueDisplayed</name>
       <value xsi:type="xsd:string">Avg_Forward_Jitter</value>
       </item>
      <item xsi:type="ns1:CIMKeyProperty">
       <name xsi:type="xsd:string">AggregateBy</name>
       <value xsi:type="xsd:string">VPN</value>
      </item>
      <item xsi:type="ns1:CIMKeyProperty">
       <name xsi:type="xsd:string">TimeGranularity</name>
        <value xsi:type="xsd:string">Weekly</value>
       </item>
      <item xsi:type="ns1:CIMKeyProperty">
        <name xsi:type="xsd:string">Timeline</name>
        <value xsi:type="xsd:dateTime">2004-1-30T00:55:38Z</value>
      </item>
     </keyProperties>
    </objectPath>
   </ns1:execReport>
  </soapenv:Body>
```

Table 5-5 Value Displayed Options for SLA Reports

Report Type	Options for ValueDisplayed
SLASummaryReport and SLASummaryCoSReport	• All • Connections • Timeouts • Connectivity • Threshold_Violations • Max_Delay • Min_Delay • Avg_Delay
SLAHTTPReport and SLAHTTPCoSReport	• All • Connectivity • Max_Delay • Threshold_Violations • DNS_RTT • DNS_Timeouts • DNS_Errors • TCP_Connection_RTT • TCP_Connection_Timeouts • Transaction_RTT • Transaction_Timeouts • Transaction_Errors
SLAJitterReport and SLAJitterCoSReport	• All • Avg_Forward_Jitter • Avg_Positive_Forward_Jitter • Avg_Negative_Forward_Jitter • Min_Forward_Jitter • Max_Forward_Jitter • Backward_Packet_Drops • Avg_Backward_Jitter • Avg_Positive_Backward_Jitter • Avg_Negative_Backward_Jitter • Min_Backward_Jitter • Max_Backward_Jitter

The output of an SLA report can an XML response or saved to a file. An output file can be in XML or CSV format. See the "Output for Reports" section on page 5-10 for more information.

Device Locking

When downloading a service configuration, the ISC provisioning engine locks the corresponding device to ensure it has dedicated access during the configuration download. By default, the back-end servers control device locking during the service provisioning process. However, you can use the API to manually lock a device to block ISC from accessing it for provisioning.

The API also supports these device locking operations:

- Locking multiple devices in batch mode
- Unlocking devices
- Viewing the status of a device lock

To manually lock a device, use an XML request with the **execMethod** operation, the **ResourceLock** object definition (className), and these parameters:

- **Action**=<choose one of the following>
  - **Lock**
  - **Unlock**
  - **Status**
- **Type**=**Device**
- Use one of these parameters to identifiy the resource to lock:
  - **Device**=<Device name>
  - **Device**=<Device ID number>
  - **JobId**=<Job ID number>

The following example shows a device locking XML request for multiple devices in batch mode.

```
   <soapenv:Body>
       <ns1:performBatchOperation>
           <actions xsi:type="ns1:CIMActionList" soapenc:arrayType="ns1:CIMAction[]">
               <action>
                   <actionName xsi:type="xsd:string">execMethod</actionName>
                   <objectPath xsi:type="ns1:CIMObjectPath">
                       <className xsi:type="xsd:string">ResourceLock</className>
                       <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
                           <item xsi:type="ns1:CIMProperty">
                               <name xsi:type="xsd:string">Action</name>
                               <value xsi:type="xsd:string">Lock</value>
                           </item>
                           <item xsi:type="ns1:CIMProperty">
                               <name xsi:type="xsd:string">Type</name>
                               <value xsi:type="xsd:string">Device</value>
                           </item>
                           <item xsi:type="ns1:CIMProperty">
                               <name xsi:type="xsd:string">Device</name>
                               <value xsi:type="xsd:string">enswosr2</value>
                           </item>
                       </properties>
                   </objectPath>
               </action>
               <action>
                   <actionName xsi:type="xsd:string">execMethod</actionName>
                   <objectPath xsi:type="ns1:CIMObjectPath">
                       <className xsi:type="xsd:string">ResourceLock</className>
```

```
                        <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">Action</name>
                                <value xsi:type="xsd:string">Status</value>
                            </item>
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">Type</name>
                                <value xsi:type="xsd:string">Device</value>
                            </item>
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">JobId</name>
                                <value xsi:type="xsd:string">0</value>
                            </item>
                        </properties>
                    </objectPath>
                </action>
                <action>
                    <actionName xsi:type="xsd:string">execMethod</actionName>
                    <objectPath xsi:type="ns1:CIMObjectPath">
                        <className xsi:type="xsd:string">ResourceLock</className>
                        <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">Action</name>
                                <value xsi:type="xsd:string">Unlock</value>
                            </item>
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">Type</name>
                                <value xsi:type="xsd:string">Device</value>
                            </item>
                            <item xsi:type="ns1:CIMProperty">
                                <name xsi:type="xsd:string">DeviceId</name>
                                <value xsi:type="xsd:string">4</value>
                            </item>
                        </properties>
                    </objectPath>
                </action>
            </actions>
        </ns1:performBatchOperation>
    </soapenv:Body>
</soapenv:Envelope>
```

The response to an XML request for device locking indicates the **Action**, **Status** and **JobId**.

- If the status value is **RUN**, your request is being processed.
- If the status is **SLEEP**, the device lock is in place.
- If the status is **FINISHED**, the lock has been removed.
- If there is a failure, a failure description is returned.

The following example is a response to a ViewResourceLock_Device XML request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message id="87855" sessiontoken="9DCB8431CB9773C285DCDBE5E1375299"
waittimeout="800000" timestamp="2004-02-27T18:42:48.051Z" wait="true" />
  </soapenv:Header>
  <soapenv:Body>
```

```
    <ns1:execMethodResponse>
      <returns xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Action</name>
          <value xsi:type="xsd:string">Status</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Status</name>
          <value xsi:type="xsd:string">SLEEP</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">JobId</name>
          <value xsi:type="xsd:string">0</value>
        </item>
      </returns>
    </ns1:execMethodResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Viewing Task Logs

The XML response to a **ViewTask** operation is an extensive list of information about the service order. This information includes:

- Information about the service order itself (**Creator**, **CreateTime**, **TaskType**, **MaxRuns**)
- The complete list of attributes in the service order
- The task log output, which includes all messages according to the level specified in the properties file

To view the logs for a specific task (for example, configuration audits, MPLS functional audits, or collection), perform the following steps:

Step 1 Record the service order **LocatorId** that is returned in the XML response.

Step 2 Run a view of the service order using ViewServiceOrder.xml, and the **LocatorId** from Step 1.

Step 3 Record the **TaskLocatorId** that is returned in the XML response.

Step 4 Run a view of the Task using ViewTask.xml. Specify **className=PersistentTask** and use the **TaskLocatorId** from Step 3.

The following example shows the tail end of the **ViewTask** XML response, which displays the task log output.

```
<xsi:type="xsd:string">view.cisco.com//opt/vpnsc/user/tmp/TaskLogs/jobLog_1064765379277_pr
ov.provdrv.ProvDrv_20</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">LogContent</name>
                      <value xsi:type="xsd:string">
     Date: 2003-09-28T10:09:39 Level: INFO Message: The argument to the ProvDrv are:
IsProvision = true
JobIdList = 3
ipsec-rekey = false
IsForceRedeploy = false
targets = []
```

```
     Date: 2003-09-28T10:09:39 Level: INFO Message: Opening repository ...
     Date: 2003-09-28T10:09:39 Level: INFO Message: Open repository succeeded
     Date: 2003-09-28T10:09:39 Level: INFO Message:  ======== Creating ProvDrvSR for
Job#3SR#3
     Date: 2003-09-28T10:09:39 Level: INFO Message: Filter to getLogicalDevices: 1
     Date: 2003-09-28T10:09:39 Level: INFO Message: getServiceElements() : ACTION -
PROVISIONING
     Date: 2003-09-28T10:09:39 Level: INFO Message: Number of logicalDevices got: 2
     Date: 2003-09-28T10:09:39 Level: INFO Message: Processing logical device 7 with
physical id 5
     Date: 2003-09-28T10:09:39 Level: INFO Message: Service blade for this device:
com.cisco.vpnsc.prov.mpls.MplsServiceBlade
     Date: 2003-09-28T10:09:39 Level: INFO Message: Create blade the first time:
com.cisco.vpnsc.prov.mpls.MplsServiceBlade
     Date: 2003-09-28T10:09:39 Level: INFO Message: created service blade
     Date: 2003-09-28T10:09:39 Level: INFO Message: returning XML_JDOM as preference
     Date: 2003-09-28T10:09:39 Level: INFO Message: Filter to generateXML: 1
     Date: 2003-09-28T10:09:39 Level: INFO Message: Generate XML: xmlPref[2] Filter
[1]
     Date: 2003-09-28T10:09:39 Level: INFO Message: getServiceElements() : ACTION -
PROVISIONING
     Date: 2003-09-28T10:09:39 Level: INFO Message: Number Of MPLS VPN Link[ 1]
     Date: 2003-09-28T10:09:40 Level: SEVERE Message: Unable to Generate input.xml
for MPLS Deployable Interface
com.cisco.vpnsc.repository.RepException: 158 : Unable to allocate IP Address from the
/32 IP Address Pool
        at
com.cisco.vpnsc.repository.servmodelaccess.MPLSServiceModelAccess.appendLink(MPLSServiceMo
delAccess.java:237)
        at
com.cisco.vpnsc.repository.servmodelaccess.MPLSServiceModelAccess.createServiceModel(MPLSS
erviceModelAccess.java:176)
        at com.cisco.vpnsc.repository.servmodelaccess.GSAM.generateXML(GSAM.java:172)
        at com.cisco.vpnsc.repository.mpls.RepMplsSR.generateXML(RepMplsSR.java:1207)
        at
com.cisco.vpnsc.prov.provdrv.ProvDrvSR.buildBladeMapAndDoBladeValidation(ProvDrvSR.java:27
2)
        at
com.cisco.vpnsc.prov.provdrv.ProvDrvSR.populateRouterLists(ProvDrvSR.java:540)
        at com.cisco.vpnsc.prov.provdrv.ProvDrv.createProvDrvSR(ProvDrv.java:2472)
        at com.cisco.vpnsc.prov.provdrv.ProvDrv.populateSRMap(ProvDrv.java:2173)
        at com.cisco.vpnsc.prov.provdrv.ProvDrv.perform(ProvDrv.java:172)
        at com.cisco.vpnsc.dist.VpnscJob.perform(VpnscJob.java:80)
        at com.cisco.vpnsc.dist.WorkerImpl.jobPerformWrapper(WorkerImpl.java:1163)
        at com.cisco.vpnsc.dist.WorkerImpl.access$000(WorkerImpl.java:31)
        at com.cisco.vpnsc.dist.WorkerImpl$JobExecutionTask.run(WorkerImpl.java:1285)
        at com.cisco.vpnsc.dist.ThreadPool$TaskThread.run(ThreadPool.java:268)

     Date: 2003-09-28T10:09:40 Level: SEVERE Message: Exception caught: null
     Date: 2003-09-28T10:09:40 Level: INFO Message: Cache input.xml with prefered
value: 2
     Date: 2003-09-28T10:09:40 Level: WARNING Message: Input XML is null!
     Date: 2003-09-28T10:09:40 Level: INFO Message: returning success for validation
     Date: 2003-09-28T10:09:40 Level: INFO Message: Processing logical device 3 with
physical id 3
     Date: 2003-09-28T10:09:40 Level: INFO Message: Service blade for this device:
com.cisco.vpnsc.prov.mpls.MplsServiceBlade
     Date: 2003-09-28T10:09:40 Level: INFO Message:  The blade
com.cisco.vpnsc.prov.mpls.MplsServiceBlade is shared by multiple types of devices.
     Date: 2003-09-28T10:09:40 Level: INFO Message: Use existing blade
com.cisco.vpnsc.prov.mpls.MplsServiceBlade
     Date: 2003-09-28T10:09:40 Level: INFO Message:  Added Router 5 to the union map.
     Date: 2003-09-28T10:09:40 Level: INFO Message: Adding logical device 7 to
Job#3SR#3 's local router list.
```

```
     Date: 2003-09-28T10:09:40 Level: INFO Message:  Add logical device 7 to
Job#3SR#3 's blade com.cisco.vpnsc.prov.mpls.MplsServiceBlade's local router list.
     Date: 2003-09-28T10:09:40 Level: INFO Message:  Added Router 3 to the union map.
     Date: 2003-09-28T10:09:40 Level: INFO Message: Adding logical device 3 to
Job#3SR#3 's local router list.
     Date: 2003-09-28T10:09:40 Level: INFO Message:  Add logical device 3 to
Job#3SR#3 's blade com.cisco.vpnsc.prov.mpls.MplsServiceBlade's local router list.
     Date: 2003-09-28T10:09:40 Level: INFO Message:
 ======== Creating ProvDrvSR succeeded for Job#3SR#3
The union router map has devices:
3 5
     Date: 2003-09-28T10:09:40 Level: INFO Message: Service Request to be processed:
Job#3SR#3 .
     Date: 2003-09-28T10:09:40 Level: INFO Message: ======== Uploading configuration
     Date: 2003-09-28T10:09:40 Level: INFO Message: Creating instance of GTL
     Date: 2003-09-28T10:09:41 Level: INFO Message: ======== Upload completed.
     Date: 2003-09-28T10:09:41 Level: INFO Message:  ======== Processing Service
Request  Job#3SR#3
     Date: 2003-09-28T10:09:41 Level: INFO Message: Entering provision method
     Date: 2003-09-28T10:09:41 Level: SEVERE Message: invalid arguments
     Date: 2003-09-28T10:09:41 Level: INFO Message:  Result.xml from
ServiceBlade[com.cisco.vpnsc.prov.mpls.MplsServiceBlade]:
null
     Date: 2003-09-28T10:09:41 Level: SEVERE Message:  Service
Blade[com.cisco.vpnsc.prov.mpls.MplsServiceBlade] returned null result for  Job#3SR#3
.
     Date: 2003-09-28T10:09:41 Level: SEVERE Message:  Job#3SR#3  skipped template
instantiation because all service blades have failed.
     Date: 2003-09-28T10:09:41 Level: SEVERE Message: Failed for all service blades.
SR Job ID 3 transitioned from INVALID to INVALID
     Date: 2003-09-28T10:09:41 Level: INFO Message: ======== Downloading
configuration.
     Date: 2003-09-28T10:09:41 Level: INFO Message: Skipping device 3, the configlet
is empty.
     Date: 2003-09-28T10:09:41 Level: INFO Message: Skipping device 5, the configlet
is empty.
     Date: 2003-09-28T10:09:41 Level: INFO Message: ======== Download completed
     Date: 2003-09-28T10:09:41 Level: INFO Message: ======== Update Service for SR
Job#3SR#3
     Date: 2003-09-28T10:09:41 Level: INFO Message: getServiceElements() : ACTION -
PROVISIONING
     Date: 2003-09-28T10:09:41 Level: INFO Message: ProvDrv is completed.</value>
                </item>
```

MPLS Provisioning

The service provider's backbone is comprised of the core provider edge (PE) device and its provider routers. An MPLS VPN consists of a set of customer sites that are interconnected through an MPLS provider core network. At each site, there are one or more customer edge (CE) devices, which attach to one or more PEs.

The Cisco IP Solution Center (ISC) provisioning engine for MPLS accesses the configuration files on both the CE and PE to compute the necessary changes to the configuration files to support the service on the PE to CE link (or PE to CLE, PE to MVRFCE to CE, or PE to MVRFCE to CLE).

This chapter describes MPLS VPN service concepts and the steps required to provision MPLS VPN services using the ISC API. The provisioning example includes the process flow from creating the inventory to auditing the service deployment.

For information on MPLS provisioning using the ISC GUI, refer to the *Cisco IP Solution Center MPLS VPN User Guide, 4.1*.

This chapter contains the following sections:

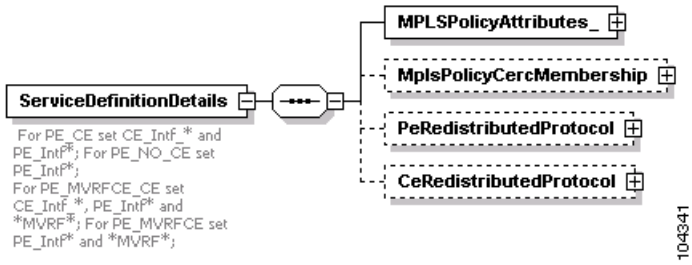
MPLS Service Definitions

An MPLS service definition defines attributes for the policy type (**MPLSPolicyAttributes**), and can include the CE routing community (CERC) membership and redistributed protocol information.

CERC membership defines the CE routing community for this policy and is represented by the VPN routing and forwarding tables (VRFs), and the redistributed protocols define the metric attributes. MPLS policy attributes are described in the MPLS Policy Attributes section.

Figure 6-1 shows the schema diagram for an MPLS service definition.

Figure 6-1 MPLS Service Definition Schema Diagram



For each service definition property, you can set an additional attribute, **editable=true,** to allow the network operator to override these attributes when creating the service request. If an attribute is set to **editable=false**, these attributes cannot be changed in the service request.

Note When a property has the additional attribute **editable=true**, all the related and child attributes are also editable.

See the following example:

```
<objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">SubType</name>
         <value xsi:type="xsd:string">PE_CE</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">PE_Intf_Type</name>
         <value xsi:type="xsd:string">GigabitEthernet</value>
        <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
       <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">PE_Intf_Desc</name>
         <value xsi:type="xsd:string">Interface Description</value>
        <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
       <item xsi:type="ns1:CIMProperty">
```

```
 <name xsi:type="xsd:string">PE_Intf_Shutdown</name>
 <value xsi:type="xsd:string">TRUE</value>
<qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
```

MPLS Policy Attributes

The MPLS policy attributes include; the policy subtype, device interfaces and encapsulation types, IP addressing schemes, routing protocols, and VPN membership information. These values are set based on the policy subtype.

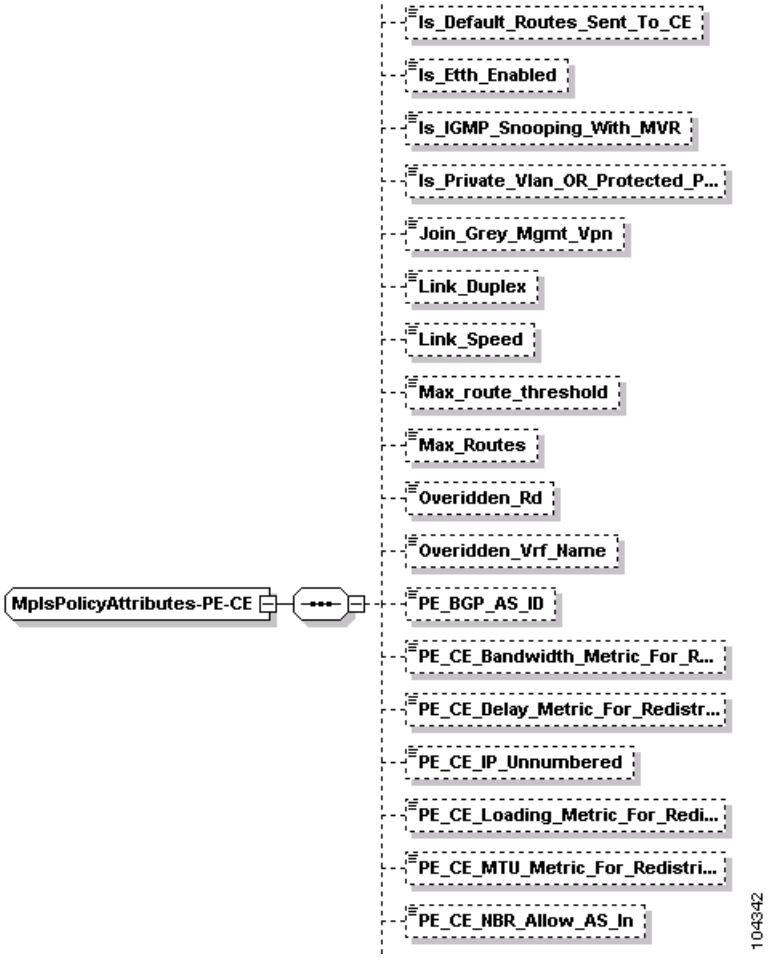
The following MPLS policy subtypes are supported:

- PE_CE—A standard MPLS policy for the PE_CE link. This is the default MPLS policy for ISC.
- PE_NO_CE—In this policy type, you specify only the PE interface information. The CE device at the customer location is not managed by ISC.
- PE_MVRFCE_CE—A Multi-VPN routing and forwarding CE (MVRFCE) network has multiple CE devices that connect to one MVRFCE device. The MVRFCE stores the VRFs for all VPNs in the customer network and connects directly with the PE device at the edge of the provider network. ISC manages the PE to MVRFCE to CE link.
- PE_MVRFCE_NO_CE—ISC manages the PE to MVRFCE link. The CE device at the customer site is not managed by ISC.

Note For all policy subtypes with no CE present, you do not declare the CE devices to be CPEs, and you do not set policy attributes for the CE devices in the service definition.

There are numerous properties that can be set for an MPLS policy. Figure 6-2 shows a partial schema diagram for the **MPLSPolicAttributes** with **SubType=PE_CE**.

Figure 6-2 MPLS Policy Attributes Schema Diagram



The following XML example shows a partial list of the properties that can be specified for the **MplsPolicyAttributes**:

```
<objectPath xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
          <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">SubType</name>
              <value xsi:type="xsd:string">PE_CE</value>
            </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">PE_CE_IP_Unnumbered</name>
                <value xsi:type="xsd:string">false</value>
                <qualifier xsi:type="ns1:CIMQualifier">
                  <name xsi:type="xsd:string">editable</name>
                  <value xsi:type="xsd:string">true</value>
                </qualifier>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">IGMP_Query_Time</name>
                <value xsi:type="xsd:string">25</value>
                <qualifier xsi:type="ns1:CIMQualifier">
```

```
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">IGMP_Mode</name>
          <value xsi:type="xsd:string">COMPATIBLE</value>
          <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Overidden_Vrf_Name</name>
          <value xsi:type="xsd:string">vrf1</value>
          <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
      <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Autopick_Vlan_ID</name>
          <value xsi:type="xsd:string">true</value>
          <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">editable</name>
            <value xsi:type="xsd:string">true</value>
          </qualifier>
        </item>
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">Extra_CE_Loopback_Required</name>
  <value xsi:type="xsd:string">FALSE</value>
 <qualifier xsi:type="ns1:CIMQualifier">
     <name xsi:type="xsd:string">editable</name>
     <value xsi:type="xsd:string">true</value>
   </qualifier>
 </item>
 <item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">Auto_Assign_IP_Address</name>
  <value xsi:type="xsd:string">TRUE</value>
 <qualifier xsi:type="ns1:CIMQualifier">
     <name xsi:type="xsd:string">editable</name>
     <value xsi:type="xsd:string">true</value>
   </qualifier>
 </item>
 <item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">IP_Address_pool_type</name>
  <value xsi:type="xsd:string">Region Pool</value>
 <qualifier xsi:type="ns1:CIMQualifier">
     <name xsi:type="xsd:string">editable</name>
     <value xsi:type="xsd:string">true</value>
   </qualifier>
 </item>
 <item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">PE_CE_Routing_Protocol</name>
  <value xsi:type="xsd:string">RIP</value>
 <qualifier xsi:type="ns1:CIMQualifier">
     <name xsi:type="xsd:string">editable</name>
     <value xsi:type="xsd:string">true</value>
   </qualifier>
 </item>
 <item xsi:type="ns1:CIMProperty">
 <name xsi:type="xsd:string">Is_Default_Routes_Sent_To_CE</name>
  <value xsi:type="xsd:string">FALSE</value>
 <qualifier xsi:type="ns1:CIMQualifier">
```

```
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
<item xsi:type="ns1:CIMProperty">
 <name xsi:type="xsd:string">Redistribute_Static</name>
 <value xsi:type="xsd:string">TRUE</value>
<qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
<item xsi:type="ns1:CIMProperty">
 <name xsi:type="xsd:string">Redistribute_Connected</name>
 <value xsi:type="xsd:string">TRUE</value>
<qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
<item xsi:type="ns1:CIMProperty">
 <name xsi:type="xsd:string">Max_Routes</name>
 <value xsi:type="xsd:string">10000</value>
<qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
```

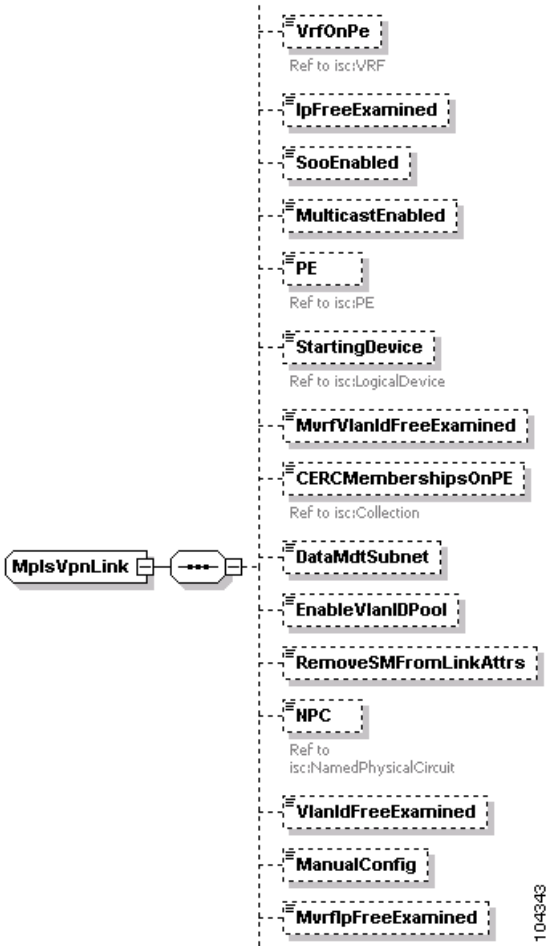
MPLS Service Requests

An MPLS service request specifies the MPLS policy (service definition) to use, the list of links, referred to as MPLS VPN links, and the link attributes. Use the link attribute settings in the service request to override any policy settings defined as editable in the service definition.

Note You can also integrate an ISC template with an MPLS service request and associate one or more templates to the CPE and PE devices. See Chapter 4, “Using Templates,” for more information.

ISC supports an extensive list of properties that can be set for MPLS VPN links. Figure 6-3 shows a partial schema diagram for the **MplsVpnLink** in a service request.

Figure 6-3 MPLS VPN Link Schema Diagram



Provisioning Example

The following sections describe the required steps for using the API to provision MPLS VPNs, and include the operation, class, and required parameters for each step.

Process Summary

This MPLS provisioning example includes the following operations:

- Create Inventory
- Create Resource Pools
- Collect Device Configurations
- Create an MPLS Policy
- Creating an MPLS Service Request

This section provides an example provisioning process using XML examples. The inventory of XML examples for the ISC API can be found at the following location:
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Create Inventory

Step 1 Create a **Provider** and **Region**.

The provider is the administrative domain of an Internet service provider (ISP), with one border gateway protocol (BGP) autonomous system (AS) number. The network owned by the provider is called the backbone network. If an ISP has two AS numbers, you must define it as two provider adrministrative domains (PADs). Each provider can contain multiple regions.

Table 6-1 Create Provider and Region

Operation	className	Required Keywords
createInstance	Provider	• Name • AsNumber
	Region	• Name • Provider

XML Examples:

- CreateProvider.xml
- CreateRegion.xml

Step 2 Create a Customer (**Organization**) and **Site**.

Table 6-2 Create Customer and Site

Operation	className	Required Keywords
createInstance	Organization	• Name
	Site	• Name • Organization

XML Examples:

- CreateOrganization.xml
- CreateSite.xml

Step 3 Create devices.

In most cases, devices are Cisco IOS routers and Catalyst switches.

Table 6-3 Create Devices

Operation	className	Required Keywords
createInstance	• CiscoRouter • CatOS	One or more of the following: • ManagementIPAddress • HostName • DomainName

XML Examples:

- CreateCiscoRouter.xml
- CreateCat.xml

Step 4 Declare devices as **PE**s and assign them to regions.

Table 6-4 Create PEs

Operation	className	Required Keywords
createInstance	PE	• Provider • Region • Role= – PE_POP – PE_CLE • Device • Interface

XML Example:

- CreatePE.xml

Step 5 Declare devices as **Cpe**s and assign them to sites.

When you declare a device as a **Cpe**, you also specify a **ManagementType** and interface information.

Table 6-5 Create CPEs

Operation	className	Required Keywords
createInstance	Cpe	• Site • Device • ManagementType

XML Example:

- CreateCpe.xml

Create Resource Pools

For MPLS provisioning, you define route targets, route distinguishers, CERCs, and VPNs.

A route target informs PEs which routes should be inserted into the appropriate VRFs. Every VPN route is tagged with one or more route targets when it is exported from a VRF and offered to other VRFs.

The IP subnets advertised by the CE routers to the PE routers are augmented with route distinguishers (RDs). The RD value must be a globally unique value to avoid conflict with other prefixes.

Table 6-6 Create Resource Pools

Operation	className	Required Keywords
createInstance	• RouteTarget • RouteDistinguisher	• Start • Size • AssocClassType • AssocClassId

Create a **VPN** and select a **CERC**.

Table 6-7 Create VPNs and CERCs

Operation	className	Required Parameters
createInstance	• VPN	• Name • CERC *or* • CreateDefaultCERC
	• CERC	• Name • Provider • SpokeRouteTarget • HubRouteTarget

A CERC can either be full mesh or hub and spoke. If you specify hub and spoke, you must specify both the **SpokeRouteTarget** and **HubRouteTarget**. For a full mesh CERC, only **SpokeRouteTarget** is required.

XML Examples:

- CreateRouteTarget.xml
- CreateRouteDistinguisher.xml
- CreateCERC.xml
- CreateVPN.xml

Collect Device Configurations

A device configuration collection is a task. This task uploads the current configuration from the device to the ISC database. The collection task is executed through a service request, and the service request is scheduled using a service order.

Table 6-8 Collect Device Configurations

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberofRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=Task • ServiceRequestDetails
	ServiceRequestDetails	• SubType=Collection • Device (or DeviceGroup) **Note** You must select at least one device or device group. • RetrieveVersion=true • RetrieveDeviceInterfaces=true

The following example is a partial XML request used to collect the configuration from device ensw4000-1:

```
<className xsi:type="xsd:string">ServiceRequestDetails</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
           <name xsi:type="xsd:string">SubType</name>
           <value xsi:type="xsd:string">COLLECTION</value>
        </item>
           <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Device</name>
          <value xsi:type="xsd:string">ensw4000-1</value>
        </item>
         <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">DeviceGroup</name>
          <value xsi:type="xsd:string">PE-Group</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">RetrieveDeviceAttributes</name>
          <value xsi:type="xsd:string">true</value> </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">RetrieveDeviceInterfaces</name>
          <value xsi:type="xsd:string">true</value> </item>
       </properties>
       </objectPath>
```

XML Example:

- CreateTaskServiceOrderCollection.xml

Create an MPLS Policy

An MPLS service policy (defined in a service definition) is a template of the parameters needed to define a service request. Once you define the policy template, it can be used by all MPLS service requests that share a common set of attributes.

An MPLS service definition consists of the **MplsPolicyAttributes**. The **MplsPolicyAttributes** define the properties specific to the policy subtype.

Table 6-9 Create an MPLS Policy

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=Mpls • ServiceDefinitionDetails
	ServiceDefinitionDetails	• MPLSPolicyAttributes • Provider or Organization **Note** If you do not specify a **Provider** or **Organization**, the service policy becomes a global policy.
	MPLSPolicyAttributes	• SubType= – PE_CE – PE_NO_CE – PE_MVRFCE_CE – PE_MVRFCE_NO_CE

XML Examples:

- CreateMPLSServiceDefn_PE_CE.xml
- CreateMPLSServiceDefn_PE_NO_CE.xml
- CreateMPLSServiceDefn_MVRF_CE.xml
- CreateMPLSServiceDefn_MVRF_NO_CE.xml

Creating an MPLS Service Request

An MPLS service request defines the service definition to use, the **MplsVpnLink** and the link attributes. The **MplsVpnLink** specifies the device interfaces involved in this service request. The link attributes contain any policy setting overrides for properties set as editable in the service definition. A service request can specify one or more MPLS VPN links.

Note Use **performBatchOperation** to group the service order and service request into one XML request.

Table 6-10 Create an MPLS Service Request

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=Mpls • ServiceRequestDetails
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=Mpls • MplsVpnLink
	MplsVpnLink	• NPC *or* • ManualConfig=true – PE – Cpe **Note** You must specify either **NPC** or **ManualConfig** to define the interfaces for the MPLS VPN links. If you use **ManualConfig**, you must also specify the interfaces (for example, **CE_Intf_Name** and **PE_Intf_Name**). • LinkTemplate (optional) **Note** See the "Templates in a Service Request" section on page 4-9.

Note The attributes **PE_Template**, **PE_Intf_Template**, **CE_Template**, and **CE_Intf_Template** allow NBI access to variables designed to hold template blobs (template blobs were used during MPLS provisioning in legacy versions of ISC).

XML Examples:

- CreateMPLSServiceOrder_PE_CE.xml
- CreateMPLSServiceOrder_PE_NO_CE.xml
- CreateMPLSServiceOrder_MVFR_CE.xml
- CreateMPLSServiceOrder_MVRF_NO_CE.xml

Auditing Service Requests

A configuration audit occurs automatically each time you deploy a service request. During this configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. An audit also verifies that there were no errors during deployment by examining the commands configured by the service request on the target devices. If the device configuration does not match what is defined in the service request, the audit flags a warning and sets the service request to a *Failed Audit* or *Lost* state.

If you do not want the configuration audit to occur, change the value for the **Audit** parameter. The **Audit** parameter supports these values:

- **Audit**—This is the default. A successfully deployed service request is automatically audited unless this flag is changed.
- **NoAudit**—Do not perform a configuration audit when the service request is deployed.
- **ForceAudit**—Perform a configuration audit even if the service request deployment is not successful.

You can use the Audit parameter with a **Create**, **Modify**, or **Decommission** service request or a **Deployment** task. See the "Service Decommission" section on page 3-10 for more information. To perform a configuration audit as a separate task, or an MPLS functional audit, see the "Tasks" section on page 3-8.

L2VPN Provisioning

To provision L2VPN using the Cisco IP Solution Center (ISC) API, you need an L2VPN service policy of a specific subtype and an L2VPN service request.

The service policy (defined in a service definition) specifies the attributes common to the end-to-end wires and attachment circuits (ACs).

The service request defines the device interfaces for each end-to-end wire connection (called link endpoints in the GUI), and can optionally override policy attributes in each end-to-end wire link and AC.

When you deploy an L2VPN service request using a service order, the attributes specified in the service definition are applied to the devices and interfaces listed in the service request, along with the attributes for each end-to-end wire link and AC.

This chapter describes L2VPN service concepts and the steps required to provision L2VPN services using the ISC API. The provisioning example includes the process flow from creating the inventory to auditing the service deployment.

For more information on L2VPN provisioning using ISC, refer to the *Cisco IP Solution Center L2VPN User Guide, 4.1*.

This chapter contains the following sections:

L2VPN Service Definitions

An L2VPN service definition specifies the policy subtype, the properties for the CPE and PE devices, and the user network interface (UNI). The properties that can be set are based on the policy subtype that is specified in the service definition.

ISC supports the following L2VPN service definition subtypes:

- EthernetEVCS (Ethernet Virtual Circuit Service)
- EthernetEVCS_NO_CE
- EthernetTLS (Transparent LAN Service)
- EthernetTLS_NO_CE
- ATM (Asynchronous Transfer Mode)
- ATM_NO_CE

- FRAME_RELAY
- FRAME_RELAY_NO_CE

Note For all service definition subtypes with NO_CE, you do not declare the CE devices to be CPEs, and you do not set policy attributes for the CE devices in the service definition.

A service definition can be shared by one or more service requests that have similar requirements. L2VPN service definitions include the following information about the end-to-end wires and attachment circuits:

- Service definition subtype (examples: EthernetEVCS or ATM_NO_CE)
- Device interface type (example: GigabitEthernet)
- VLAN IDs
- UNI Port Security
- Protocols (examples: CDP, VTP)

Note For each service definition property, you can set an additional attribute, **editable=true,** to allow the network operator to override these attributes when creating the service request. If an attribute is set to **editable=false**, these attributes cannot be changed in the service request.

L2VPN Service Requests

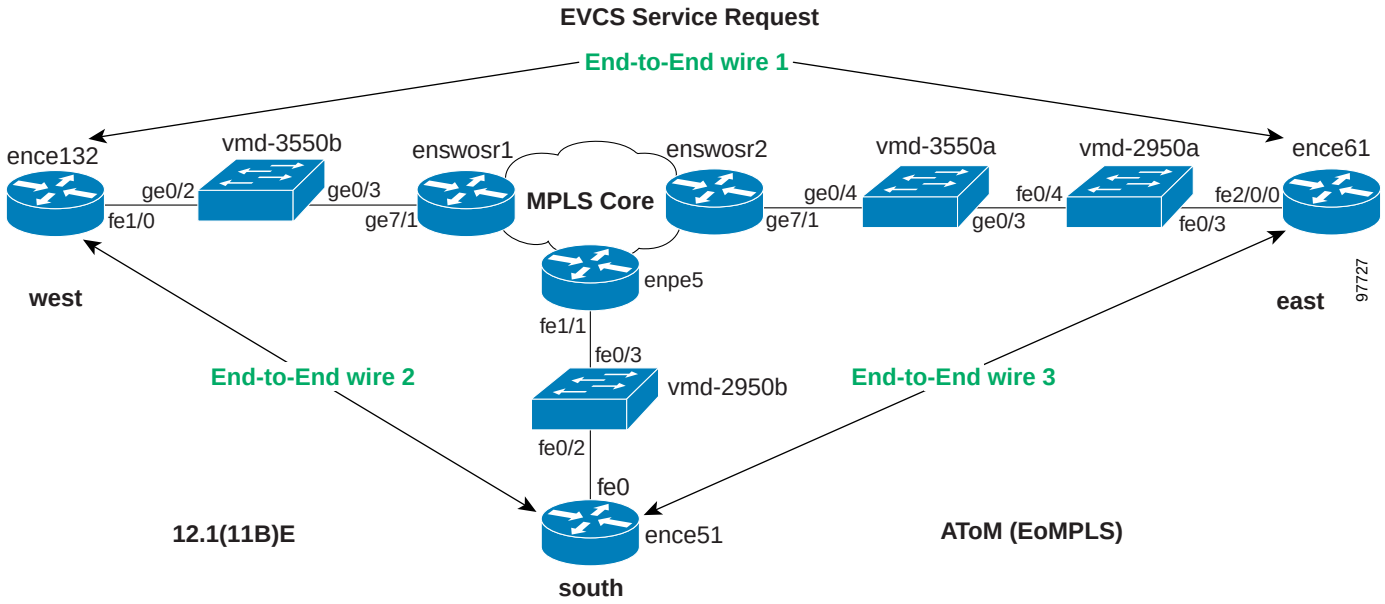
An L2VPN service request specifies the service definition, assigns device interfaces for the end-to-end wire connections, and the attachment circuit details.

End-To-End Wires

An end-to-end wire is the link from one endpoint through the service provider cloud to another endpoint. In most cases, these links are from CE to CE. An attachment circuit is the link from the CE to the PE. If there is not CE present, the attachment circuit link is from PE-CLE to PE-CLE.

In the **EthernetEVCS** network example shown in Figure 7-1, End-to-End wire1 is shown from ence132 to ence61. The two attachment circuits are the links from ence132 to enswosr1 and from ence61 to enswosr2.

Figure 7-1 End to End Wire Network Example



The number of end-to-end wires and attachment circuits that can be created are based on the policy subtype.

There can be 1 or 2 **AttachmentCircuits** for every **EndToEndWire** for the following policy subtypes:

- EthernetEVCS
- EthernetEVCS_NO_CE
- EthernetTLS
- EthernetTLS_NO_CE

There are 2 **AttachmentCircuits** for every **EndToEndWire** for the following policy subtypes:

- FRAME_RELAY
- FRAME_RELAY_NO_CE
- ATM
- ATM_NO_CE

Provisioning Example

This section describes the required steps for using the API to provision L2VPN, and includes the operation, class, and required parameters for each step.

Process Summary

In this L2VPN provisioning example, the following steps are listed:

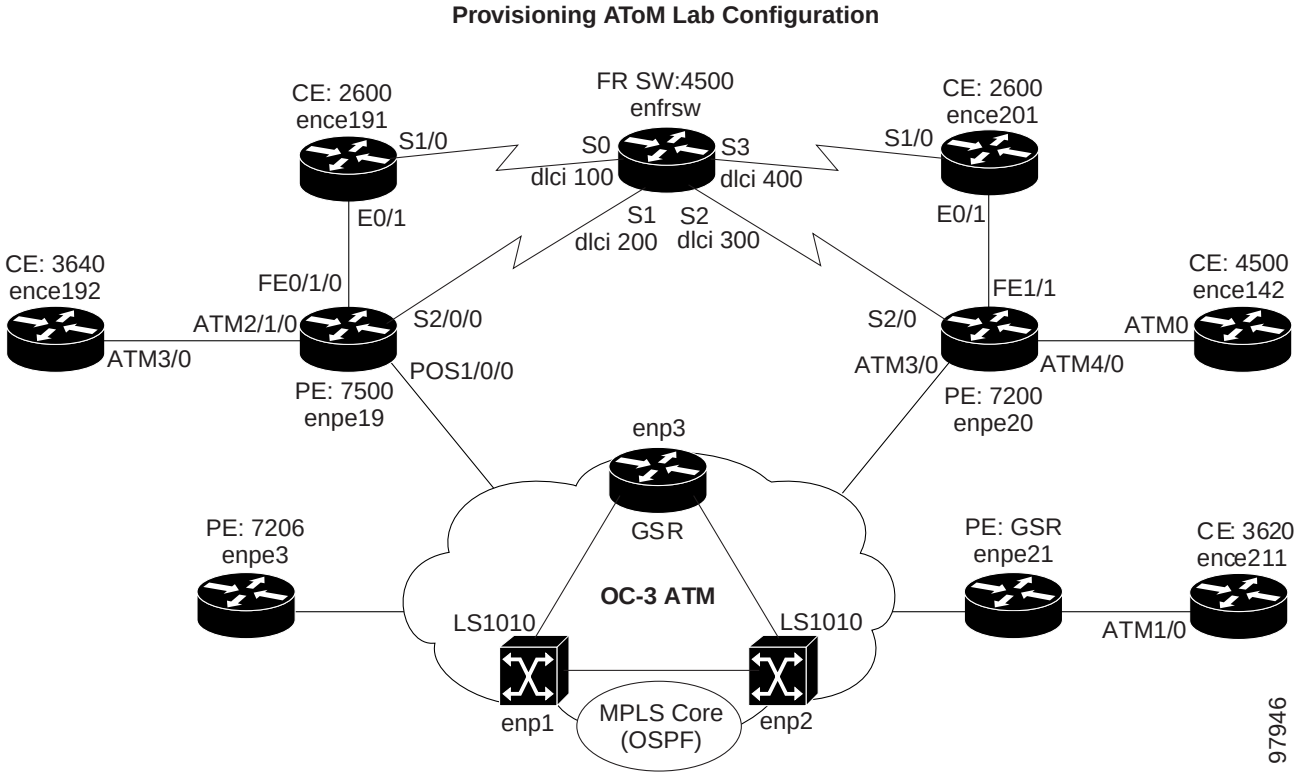
- Create device groups (optional)
- Create devices

- Collect device configurations
- Create provider
- Create regions
- Declare devices as PEs
- Create access domains
- Create customer
- Create sites
- Declare devices as CPEs (not required for No_CE service subtypes)
- Create named physical circuits
- Create VLAN ID pool
- Create VC ID pool
- Create VPN
- Create the L2VPN service definition (policy)
- Create the L2VPN service request

Note For clarity, this provisioning process shows each step as a separate XML request. Many of these steps can be combined using **performBatchOperations**.

The provisioning example in this section is based on the partial network diagram shown in Figure 7-2.

Figure 7-2 L2VPN Provisioning Network Example



Prerequisites

For security reasons, ISC requires the virtual terminal protocol (VTP) to be configured in transparent mode on all switches involved in Ethernet Relay Service (ERS) or Ethernet Wire Service (EWS) before provisioning L2VPN service requests.

To set the VTP mode, enter the following Cisco IOS commands:

```
configure terminal
vtp mode transparent
```

Enter the following Cisco IOS command to verify that the VTP has changed to transparent mode:

```
Show vtp status
```

RBAC

ISC uses a Cisco role-based access control (RBAC) product for user login and logoff. These user roles and permissions are set up using the GUI.

When you establish an API session, you are given a session token during the login. For each API XML request, the session token is verified against the RBAC processor to ensure that the API user has permissions for that operation. If the user does not have permissions, the API returns an error.

Refer to the *Cisco IP Solution Center Infrastructure Reference, 4.1* for information on setting up user roles and permissions.

Provisioning Process

This section provides an example provisioning process using XML examples. The inventory of XML examples for the ISC API can be found at the following location:
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Step 1 Create device groups (optional).

Table 7-1 Create Device Group

Operation	className	Required Parameters
createInstance	DeviceGroup	Name

In this example, one device group is created for the customer, and one for the provider.

- CreateDeviceGroup_ProvDev.xml
- CreateDeviceGroup_CustDev.xml

XML Examples:

CreateDeviceGroup.xml

Tip If you plan to create device groups, create the empty device groups before you create the devices. As you create each device, add the associated device group name as a key property in the create device XML request.

In the following example, the device group (CustDev) is added as a key property when creating the device **CiscoRouter**:

```
<ns1:createInstance>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">CiscoRouter</className>
           <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
          <item xsi:type="ns1:CIMProperty">
               <name xsi:type="xsd:string">DeviceGroup</name>
               <value xsi:type="xsd:string">CustDev</value>
            </item>
          <item xsi:type="ns1:CIMProperty">
               <name xsi:type="xsd:string">CfgUpDnldMech</name>
               <value xsi:type="xsd:string">DEFAULT</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">TransportMechanism</name>
              <value xsi:type="xsd:string">DEFAULT</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">Password</name>
              <value xsi:type="xsd:string">vpnsc</value>
            </item>
```

Step 2 Create devices.

Every network element that ISC manages must be defined as a device in the system. An element is any device from which ISC can collect information. In most cases, devices are Cisco IOS routers and Catalyst switches.

Table 7-2 Create Devices

Operation	className	Required Parameters
createInstance	• CiscoRouter • CatOS	One or more of the following: • ManagementIPAddress • HostName • DomainName

In this example, an XML request is created for each device in the CPE to PE link, as shown in Figure 7-2.

- CreateDevice_enpe20.xml
- CreateDevice_enpe21.xml
- CreateDevice_ence142.xml
- CreateDevice_ence211.xml

XML Examples:

- CreateCiscoRouter.xml
- CreateCat.xml

Step 3 Collect device configurations.

A device configuration collection is a task. This task uploads the current configuration from the device to the ISC database. The collection task is executed through a service request, and the service request is scheduled through a service order.

Table 7-3 Collect Device Configurations

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberofRequests • ServiceRequest

Table 7-3 Collect Device Configurations (continued)

Operation	className	Required Parameters
	ServiceRequest	• RequestName • Type=Task • ServiceRequestDetails
	ServiceRequestDetails	• SubType=COLLECTION • Device (or DeviceGroup) **Note** You must select at least one device or device group. • RetrieveVersion=true • RetrieveDeviceInterfaces=true

In this example, device collection is divided into two separate tasks. One task performs a configuration collection on the devices in the customer device group, and the second task is for the provider device group. (Device groups were created in Step 1.)

- CreateTaskServiceOrderCollection1.xml
- CreateTaskServiceOrderCollection2.xml

Note To perform a collection on a device group, specify the **DeviceGroup** keyword in the **ServiceRequestDetails**. (For example, **DeviceGroup=Group_CustDev**, **DeviceGroup=Group_ProvDev**).

XML Example:

- CreateTaskServiceOrderCollection.xml

Step 4 Create Provider.

The provider is the administrative domain of an ISP, with one BGP autonomous system (AS) number. The network owned by the provider is called the backbone network. If an ISP has two AS numbers, you must define it as two provider administrative domains.

Table 7-4 Create Provider

Operation	className	Required Keywords
createInstance	Provider	• Name • AsNumber

XML Example:

- CreateProvider.xml

Step 5 Create Regions.

Each provider can contain multiple regions. In this example, an XML request is created for each region.

- CreateRegion1.xml
- CreateRegion2.xml

Table 7-5 Create Region

Operation	className	Required Keywords
createInstance	Region	• Name • Provider

XML Example:

- CreateRegion.xml

Step 6 Declare devices as PEs.

The XML request that assigns a PE role (PE-POP or PE-CLE) to a device is also used to:

- Assign PE devices to Regions/Provider
- Specify PE interface information

Table 7-6 Create PEs

Operation	className	Required Keywords
createInstance	PE	• Provider • Region • Role= – PE_CLE – PE_POP • Device • Interface

In this example, an XML request is created for each device to be declared as a PE. Using Figure 7-2 for reference, enpe21=PE1 and enpe20=PE2.

- CreatePE1.xml
- CreatePE2.xml

XML Example:

- CreatePE.xml

Step 7 Create Access Domains.

Create an access domain for Ethernet-based services where ISC automatically assigns a VLAN for the link from the VLAN pool. Select all PE-POP devices to be associated with this domain, and later in the process, when VLAN pools are created for an Access Domain, the PE-POP is automatically assigned a VLAN ID.

Table 7-7 Create Access Domains

Operation	className	Required Keywords
createInstance	AccessDomain	• Name • Provider • PE (Role must be PE_POP)

XML Example:

- CreateAccessDomain.xml

Step 8 Create Customer.

A customer is a requestor of VPN services. Each customer can contain multiple customer sites. Each site belongs to only one customer and can contain many CPEs.

Table 7-8 Create Customer

Operation	className	Required Keywords
createInstance	Organization	• Name

XML Examples:

- CreateOrganization.xml

Step 9 Create Sites.

Create sites and assign customers (**Organizations**) to them. In this example, an XML request is created for each site.

- CreateSite1.xml
- CreateSite2.xml

Table 7-9 Create Sites

Operation	className	Required Keywords
createInstance	Site	• Name • Organization

XML Examples:

- CreateSite.xml

Step 10 Declare devices as CPEs.

The XML request that assigns a CPE to a site is also used to specify:

- The management type.
- CE interface information. If no CE is present, specify the UNI (PE-CLE or PE-POP) interface information.

Table 7-10 Create CPE Devices

Operation	className	Required Keywords
createInstance	Cpe	• Site • Device • ManagementType

In this example, an XML request is created for each device you want specified as a CPE. Using the network diagram for reference, ence142=Cpe1 and ence211=Cpe2.

- CreateCpe1.xml
- CreateCpe2.xml

XML Example:

- CreateCpe.xml

Step 11 Create Named Physical Circuits.

Create an NPC for each physical link in the CPE to PE-POP end-to-end wire. If there are intermediate devices, those links must also be added to the NPC using **PhysicalLink**.

Note When creating an NPC, you must specify the CPE as the source device and the PE-POP as the destination device. If there are intermediate devices, such as PE-CLEs, the source and destination devices must follow the direction of the CPE to PE-POP wire. If no CE is present, the source device interface is the UNI.

Table 7-11 Create NPCs

Operation	className	Required Parameters
createInstance	NamedPhysicalCircut	PhysicalLink
	PhysicalLink	• SrcDevice • DestDevice • SrcIfName • DestIfName

In this example, create an XML request for each CPE to PE-POP link. Using Figure 7-2 for reference, ence142 to enpe20=NPC1 and from ence211 to enpe21=NPC2.

- CreateNPC1.xml
- CreateNPC2.xml

Optionally, you can create one XML request for the **NamedPhysicalCircuit** and include multiple **PhysicalLinks** as shown in the following example:

```
<ns1:createInstance>
     <objectPath xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">NamedPhysicalCircuit</className>
      <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
      </properties>
      <objectPath xsi:type="ns1:CIMObjectPath">
           <className xsi:type="xsd:string">PhysicalLink</className>
           <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">SrcDevice</name>
             <value xsi:type="xsd:string">Device1</value> </item>
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">DestDevice</name>
             <value xsi:type="xsd:string">Device2</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">SrcIfName</name>
              <value xsi:type="xsd:string">Intf1/0</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">DestIfName</name>
              <value xsi:type="xsd:string">Intf2/1</value> </item>
            </properties>
      <objectPath xsi:type="ns1:CIMObjectPath">
           <className xsi:type="xsd:string">PhysicalLink</className>
           <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">SrcDevice</name>
             <value xsi:type="xsd:string">Device3</value> </item>
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">DestDevice</name>
             <value xsi:type="xsd:string">Device5</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">SrcIfName</name>
              <value xsi:type="xsd:string">Intf3/0</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">DestIfName</name>
              <value xsi:type="xsd:string">Intf5/1</value> </item>
            </properties>
        </objectPath>
       </objectPath>
   </ns1:createInstance>
```

XML Examples:

- CreateNamedPhysicalCircuit.xml
- CreateNamedPhysicalCircuitRing.xml—Use this example if there is a Ring topology configuration on the PE-CLEs.
- CreateNamedPhysicalCircuitRingExisting.xml—Use this example to reference an NPC ring that has already been created.

Step 12 Create VLAN ID pool.

Create a VLAN ID pool, specify a range, and associate it to an access domain to manually enter the parameters for a VLAN pool. To have ISC automatically assign VLANs to end-to-end wire links, specify the **AutoPickVlanId** keyword in the service definition (see Step 15).

Table 7-12 Create VLAN Pool

Operation	className	Required Keywords
createInstance	VlanIdPool	• Start • Size • AssocClassType • AssocClassId

XML Example:

- CreateVlanIdPool.xml

Step 13 Create a VC ID pool. A VC ID pool is global and not associated with a provider or organization.

Table 7-13 Create VC ID Pool

Operation	className	Required Keywords
createInstance	VcIdPool	• Start • Size

XML Example:

- CreateVcIdPool.xml

Step 14 Create a VPN.

A VPN in an L2VPN network is only a name used to group L2VPN links.

Table 7-14 Create VPNs

Operation	className	Required Parameters
createInstance	VPN	• Name

XML Example:

- CreateVPN.xml

Step 15 Create the L2VPN service definition (policy).

A service definition is a template of the parameters needed to define an L2VPN service request. Once you define the policy template, it can be used by all L2VPN service requests that share a common set of attributes.

Certain properties in the service definition can set an additional attribute, **editable=true**. This allows the service request creator to change the values on specific policy attributes. If the property is set to **editable=false**, the service request creator cannot change the policy attributes. See the following example:

```
<objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
            <properties xsi:type="ns1:CIMPropertyList"
```

```
          soapenc:arrayType="ns1:CIMProperty[]">
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">SubType</name>
  <value xsi:type="xsd:string">ATM</value>
</item>
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">PE_Encap</name>
  <value xsi:type="xsd:string">AAL0</value>
  <qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">CE_Intf_Type</name>
  <value xsi:type="xsd:string">Switch</value>
  <qualifier xsi:type="ns1:CIMQualifier">
    <name xsi:type="xsd:string">editable</name>
    <value xsi:type="xsd:string">true</value>
  </qualifier>
</item>
```

In this example, a service definition is created, with a **SubType=ATM**, and policy attribute values for the Cpe, PE, and UNI with the **ServiceDefinitionDetails**.

- CreateL2VPNServiceDefn_ATM.xml

Table 7-15 Create a Service Policy

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=L2Vpn • ServiceDefinitionDetails
	ServiceDefinitionDetails	• SubType= – EthernetEVCS – EthernetEVCS_NO_CE – EthernetTLS – EthernetTLS_NO_CE – FRAME_RELAY – FRAME_RELAY_NO_CE – ATM – ATM_NO_CE • Provider or Organization **Note** If you do not specify a **Provider** or **Organization**, the service policy is global.

Note If you set the **AutoPickVlanId** keyword in the **ServiceDefinitionDetails**, be sure that an access domain is attached to the PE that the PE-CLE (switch) is connected to, and a VLAN pool is assigned to the access domain.

XML Examples:

- CreateL2VPNServiceDefn_EVCS.xml
- CreateL2VPNServiceDefn_EthernetEVCS_NO_CE.xml
- CreateL2VPNServiceDefn_EthernetTLS.xml
- CreateL2VPNServiceDefn_EthernetTLS_NO_CE.xml
- CreateL2VPNServiceDefn_ATM.xml
- CreateL2VPNServiceDefn_ATM_NO_CE.xml
- CreateL2VPNServiceDefn_FRAME_RELAY.xml
- CreateL2VPNServiceDefn_FRAME_RELAY_NO_CE.xml

Step 16 Create the L2VPN service request.

An L2VPN service request consists of one or more end-to-end wires, connecting various sites in a point-to-point topology. When you create a service request, you enter several parameters, including the service definition to use, the specific interfaces on the CPE (or UNI) and PE devices, routing protocol information, and IP addressing information.

In this example, an L2VPN service request is created for an ATM network with the CE present. The service request is deployed through a service order. The **ServiceRequestDetails** specify the attributes for the end-to-end wires and attachment circuits.

Table 7-16 Create a Service Request

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=L2Vpn • ServiceRequestDetails
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=L2Vpn • EntoEndWire • VPN
	EndtoEndWire	• AttachmentCircuit
	AttachmentCircuit	• ACAttrs – LinkTemplate (optional) **Note** See the "Templates in a Service Request" section on page 4-9.

Tip Record the **LocatorId** value that is returned for the service request. The locator ID is required to perform a configuration audit of the service request.

XML Examples:

- CreateL2VPNServiceOrder_EVCS.xml
- CreateL2VPNServiceOrder_EVCS_NO_CE.xml
- CreateL2VPNServiceOrder_EthernetTLS.xml
- CreateL2VPNServiceOrder_EthernetTLS_NO_CE.xml
- CreateL2VPNServiceOrder_ATM.xml
- CreateL2VPNServiceOrder_ATM_NO_CE.xml
- CreateL2VPNServiceOrder_FRAME_RELAY.xml
- CreateL2VPNServiceOrder_FRAME_RELAY_NO_CE.xml

Auditing Service Requests

A configuration audit occurs automatically each time you deploy a service request. During this configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. An audit also verifies that there were no errors during deployment by examining the commands configured by the service request on the target devices. If the device configuration does not match what is defined in the service request, the audit flags a warning and sets the service request to a *Failed Audit* or *Lost* state.

If you do not want the configuration audit to occur, change the value for the **Audit** parameter. The **Audit** parameter supports these values:

- **Audit**—This is the default. A successfully deployed service request is automatically audited unless this flag is changed.
- **NoAudit**—Do not perform a configuration audit when the service request is deployed.
- **ForceAudit**—Perform a configuration audit even if the service request deployment is not successful.

You can use the Audit parameter with a **Create**, **Modify**, or **Decommission** service request or a **Deployment** task. See the "Service Decommission" section on page 3-10 for more information. To perform a configuration audit as a separate task, see the "Configuration Audit" section on page 3-11.

VPLS Provisioning

The Cisco IP Solution Center (ISC) supports layer 2 provisioning with layer 2 Virtual Private Network (L2VPN) services and Virtual Private LAN services (VPLS). VPLS services are multipoint (L2VPN are point-to-point) and include Ethernet services over a Multiprotocol Label Switching (MPLS) core or over an Ethernet core.

With an MPLS-based provider core, the PE devices forward customer Ethernet traffic through the core using a VPLS VPN. Multiple attachment circuits are joined together by the provider core and simulate a virtual bridge that connects all the attachment circuits together.

With an Ethernet-based provider core, the PE devices connect two or more customer devices using 802.1Q-in-Q tag-stacking technology, which encapsulates traffic from multiple VLANs from one customer with a single service provider tag. All connections within the VPLS VPN are peers and have direct communications, like a distributed switch.

For each of the core types, ISC supports Ethernet relay service (ERS) and multipoint Ethernet wire service (EWS).

- ERS—The PE device forwards all Ethernet packets with a particular VLAN tag received from an attachment circuit (excluding bridge protocol data units (BPDUs)), to another attachment circuit.
- EWS—The PE device forwards all Ethernet packets received from an attachment circuit (including tagged (**DOT1Q**), untagged (**DEFAULT**), and BPDUs), to either another attachment circuit or to all attachment circuits.

To provision VPLS using the ISC API, you need a VPLS service definition and a VPLS service request. The service definition specifies the core type, policy subtype, and common device properties. The service request defines the service definition to use, VPNs, attributes for each interface in the VPLS link, and template information.

This chapter describes VPLS service concepts and the steps required to provision VPLS services using the ISC API. The provisioning example includes all steps from creating the inventory to auditing the service deployment.

For more information on VPLS provisioning using ISC, refer to the *Cisco IP Solution Center L2VPN User Guide, 4.1*.

This chapter contains the following sections:

VPLS Service Definitions

A VPLS service definition specifies the core type, policy subtype, device properties, and the attributes common to all attachment circuits.

ISC supports the following:

- Policy subtypes:
  - VPLS_EWS
  - VPLS_EWS_NO_CE
  - VPLS_ERS
  - VPLS_ERS_NO_CE

Note For all service definition subtypes with NO_CE, you do not declare the CE devices to be CPEs and you set policy attributes for the PE and the UNI (the PE-POP or PE-CLE).

- Core types:
  - MPLS—provider core network is MPLS-enabled
  - Ethernet—provider core network uses Ethernet switches
- Information about the attachment circuits:
  - Device interface type (example: GigabitEthernet)
  - VLAN IDs
  - UNI Port Security
  - Protocols (examples: CDP, VTP)

The properties to set for the PE and CE/UNI are based on the core type and service definition subtype. For example, if the policy subtype is:

- **VPLS_ERS** for an Ethernet core, you specify the CE interface type, format and encapsulation type, and whether to filter BPDU.
- **VPLS_EWS_NO_CE** for an MPLS core, you specify the PE/CLE interface type and format, protocol tunneling, and system MTU.

Note For each service definition property, you can set an additional attribute, **editable=true,** to allow the network operator to override these attributes when creating the service request. If an attribute is set to **editable=false**, these attributes cannot be changed in the service request.

See the following example for a **VPLS_ERS** service definition:

```
<soapenv:Body>
    <ns1:createInstance>
      <objectPath xsi:type="ns1:CIMObjectPath">
        <className xsi:type="xsd:string">ServiceDefinition</className>
        <properties xsi:type="ns1:CIMPropertyList"
                soapenc:arrayType="ns1:CIMProperty[]">
          <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">Name</name>
            <value xsi:type="xsd:string">VplsPolicyERS</value>
          </item>
          <item xsi:type="ns1:CIMProperty">
```

```
  <name xsi:type="xsd:string">Type</name>
  <value xsi:type="xsd:string">Vpls</value>
</item>
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">Organization</name>
  <value xsi:type="xsd:string">NbiCustomer</value>
</item>
</properties>
<objectPath xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
  <properties xsi:type="ns1:CIMPropertyList"
              soapenc:arrayType="ns1:CIMProperty[]">
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">SubType</name>
      <value xsi:type="xsd:string">VPLS_ERS</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">Core_Type</name>
      <value xsi:type="xsd:string">MPLS</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">editable</name>
        <value xsi:type="xsd:string">true</value>
      </qualifier>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">CE_Intf_Type</name>
      <value xsi:type="xsd:string">GigabitEthernet</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">editable</name>
        <value xsi:type="xsd:string">true</value>
      </qualifier>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">CE_Encap</name>
      <value xsi:type="xsd:string">DOT1Q</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">editable</name>
        <value xsi:type="xsd:string">true</value>
      </qualifier>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">PE_Encap</name>
      <value xsi:type="xsd:string">DOT1Q</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">editable</name>
        <value xsi:type="xsd:string">true</value>
      </qualifier>
```

VPLS Service Requests

An VPLS service request defines the service definition and VPN to use, assigns interfaces and attributes for each attachment circuit (**VplsLink**), and applies template information.

Note The attachment circuit, or **VplsLink**, defines the layer 2 path from the CE to the PE, including any intermediate devices. You provision attachment circuits on the PE-facing port of the CE, on all ports of intermediate devices, and on the CE-facing port on the PE.

When you deploy a VPLS service request using a service order, the attributes specified in the service definition are applied to the devices and interfaces defined in the service request, along with the link attributes for each attachment circuit.

The service request link attributes include any parameters marked as editable in the service definition and link parameters specific to each attachment circuit. The following XML example shows a partial list of the properties that can be specified for the attachment circuit **LinkAttrs**.

```
<objectPath xsi:type="ns1:CIMObjectPath">
                  <className xsi:type="xsd:string">LinkAttrs</className>
                  <properties xsi:type="ns1:CIMPropertyList"
                     soapenc:arrayType="ns1:CIMProperty[]">
                   <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">CE_Intf_Name</name>
                      <value xsi:type="xsd:string">GigabitEthernet0/1</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">PE_Intf_Name</name>
                      <value xsi:type="xsd:string">GigabitEthernet1/1</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Disable_CDP</name>
                      <value xsi:type="xsd:string">false</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Duplex</name>
                      <value xsi:type="xsd:string">Half</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Shutdown</name>
                      <value xsi:type="xsd:string">true</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Speed</name>
                      <value xsi:type="xsd:string">100</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Port_Security</name>
                      <value xsi:type="xsd:string">true</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Max_Mac_Address</name>
                      <value xsi:type="xsd:string">13</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Violation_Action</name>
                      <value xsi:type="xsd:string">PROTECT</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Aging</name>
                      <value xsi:type="xsd:string">234</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Uni_Protocol_Tunnelling</name>
                      <value xsi:type="xsd:string">true</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Tunnel_Stp_Threshold</name>
                      <value xsi:type="xsd:string">3001</value>
                    </item>
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">Tunnel_Cdp_Threshold</name>
                      <value xsi:type="xsd:string">1001</value>
```

```
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Tunnel_Vtp_Threshold</name>
    <value xsi:type="xsd:string">2001</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Tunnel_Stp_Enable</name>
    <value xsi:type="xsd:string">true</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Tunnel_Vtp_Enable</name>
    <value xsi:type="xsd:string">true</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Tunnel_Recovery_Interval</name>
    <value xsi:type="xsd:string">4001</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Tunnel_Cdp_Enable</name>
    <value xsi:type="xsd:string">true</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Autopick_Vlan_ID</name>
    <value xsi:type="xsd:string">true</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">System_MTU</name>
    <value xsi:type="xsd:string">1522</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">PE_Intf_Desc</name>
    <value xsi:type="xsd:string">Pe Intf Desc</value>
  </item>
</properties>
 <objectPath xsi:type="ns1:CIMObjectPath">
<className xsi:type="xsd:string">LinkTemplate</className>
<properties xsi:type="ns1:CIMPropertyList"
                              soapenc:arrayType="ns1:CIMProperty[]">
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">LogicalDevice</name>
    <value xsi:type="xsd:string">ensw3550-1</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">DatafilePath</name>
    <value xsi:type="xsd:string">/nbi/AccessList</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">TemplateActive</name>
    <value xsi:type="xsd:string">true</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">TemplateAction</name>
    <value xsi:type="xsd:string">APPEND</value>
  </item>
</properties>
```

Provisioning Example

This section describes the process for using the API to provision VPLS, and includes the operation, object definition (className), and parameter definitions.

Process Summary

This VPLS provisioning example uses the following processes:

- Create device groups (optional)
- Create devices
- Collect device configurations
- Create provider
- Create regions
- Declare devices as PEs
- Create access domains and assign PEs to them
- Create customer
- Create sites
- Declare devices as CPEs (not required for service types with no CE present)
- Assign CPE devices to sites
- Create named physical circuits (not required if **ManualConfig=true**)
- Create VLAN ID pool
- Create VC ID pool
- Create VPN
- Create VPLS service definition (policy)
- Create VPLS service request

Prerequisites

For security reasons, ISC requires the virtual terminal protocol (VTP) to be configured in transparent mode on all switches involved in Ethernet Relay Service (ERS) or Ethernet Wire Service (EWS) before provisioning VPLS service requests.

To set the VTP mode, enter the following Cisco IOS commands:

```
configure terminal
vtp mode transparent
```

Enter the following Cisco IOS command to verify that the VTP has changed to transparent mode:

```
Show vtp status
```

RBAC

ISC uses a Cisco role-based access control (RBAC) product for user login and logoff. These user roles and permissions are set up using the GUI.

When you establish an API session, you are given a session token during the login. For each API XML request, the session token is verified against the RBAC processor to ensure that the API user has permissions for that operation. If the user does not have permissions, the API returns an error.

Refer to the *Cisco IP Solution Center Infrastructure Reference, 4.1* for information on setting up user roles and permissions.

Provisioning Process

This section describes the process for provisioning VPLS using XML examples.

The complete list of XML examples for VPLS are located at:
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Note For clarity, this provisioning process shows each step as a separate XML request. Many of these steps can be combined using **performBatchOperations**.

Step 1 Create device groups (optional).

Table 8-1 Create Device Group

Operation	className	Required Parameters
createInstance	DeviceGroup	Name

XML Examples:

- CreateDeviceGroup.xml

Tip If you plan to create device groups, create the empty device groups before you create the devices. As you create each device, add the associated device group name as a key property in the create device XML request.

In the following example, the device group (CustDev) is added as a key property when creating the device **CiscoRouter**:

```
<ns1:createInstance>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">CiscoRouter</className>
            <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
           <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">DeviceGroup</name>
                <value xsi:type="xsd:string">CustDev</value>
              </item>
           <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">CfgUpDnldMech</name>
                <value xsi:type="xsd:string">DEFAULT</value>
              </item>
```

```
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">TransportMechanism</name>
  <value xsi:type="xsd:string">DEFAULT</value>
</item>
<item xsi:type="ns1:CIMProperty">
  <name xsi:type="xsd:string">Password</name>
  <value xsi:type="xsd:string">vpnsc</value>
</item>
```

Step 2 Create devices.

Every network element that ISC manages must be defined as a device in the system. An element is any device from which ISC can collect configuration information.

Table 8-2 Create Devices

Operation	className	Required Parameters
createInstance	• CiscoRouter • CatOS **Note** If the service definition policy subtype is **VPLS_ERS**, the CE must be a **CiscoRouter**.	One or more of the following: • ManagementIPAddress • HostName • DomainName

XML Examples:

- CreateCiscoRouter.xml
- CreateCat.xml

Step 3 Collect device configurations.

A device configuration collection is a task. This task uploads the current configuration from the device to the ISC database. The collection task is executed through a service request, and the service request is scheduled through a service order.

Table 8-3 Collect Device Configurations

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberofRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=Task • ServiceRequestDetails
	ServiceRequestDetails	• SubType=COLLECTION • Device (or DeviceGroup) **Note** You must select at least one device or device group. • RetreiveVersion=true • RetreiveDeviceInterfaces=true

XML Examples:

- CreateTaskServiceOrderCollection.xml

Step 4 Create a provider.

The provider is the administrative domain of an ISP, with one BGP autonomous system (AS) number. The network owned by the provider is called the backbone network. If an ISP has two AS numbers, you must define it as two provider administrative domains.

Table 8-4 Create a Provider

Operation	className	Required Parameters
createInstance	Provider	• Name • AsNumber

XML Examples:

CreateProvider.xml

Step 5 Create regions.

Each provider can contain multiple regions.

Table 8-5 Create Regions

Operation	className	Required Parameters
createInstance	Region	• Name • Provider

XML Examples:

CreateRegion.xml

Step 6 Declare devices as PEs.

The XML request that assigns a PE role to a device is also used to:

- Assign PE devices to Regions/Provider
- Specify interface information

Table 8-6 Create PE Devices

Operation	className	Required Parameters
createInstance	PE	• Provider • Region • Role= – PE_CLE – PE_POP • Device • Interface

XML Examples:

CreatePE.xml

Step 7 Create access domains.

ISC assigns a VLAN ID to the attachment circuit from the VLAN ID pool. Select all PE-POP devices to be associated with this domain, and later in the process, when VLAN ID pools are created, the PE-POP is automatically assigned a VLAN ID.

Note If provisioning for an Ethernet provider core, all PE devices must be in the same access domain and a single VLAN ID is used for the entire VPLS VPN. If provisioning for an MPLS provider core, the PE devices can be in different access domains.

Table 8-7 Create Access Domains

Operation	className	Required Parameters
createInstance	AccessDomain	• Name • Provider • PE (Role must be PE_POP)

XML Examples:

- CreateAccessDomain.xml

Step 8 Create a customer.

A customer is a requestor of VPN services. Each customer can contain multiple customer sites. Each site belongs to only one customer and can contain multiple CPEs.

Table 8-8 Create Organization

Operation	className	Required Parameters
createInstance	Organization	• Name

XML Examples:

- CreateOrganization.xml

Step 9 Create sites and assign customers (**Organizations**) to them.

Table 8-9 Create Sites

Operation	className	Required Parameters
createInstance	Site	• Name • Organization

XML Examples:

- CreateSite.xml

Step 10 Declare devices as CPEs. This step is not required for service subtypes with no CE present.

Table 8-10 Create CPE Devices

Operation	className	Required Parameters
createInstance	Cpe	• Site • Device • ManagementType

XML Examples:

- CreateCpe.xml

Step 11 Create Named Physical Circuits (NPCs). This step is not required if you plan to manually configure the physical links in the VPLS service request (Step 16).

Create an NPC for each attachment circuit (CE/UNI to PE-POP link). If there are intermediate devices, those links must also be added to the NPC using **PhysicalLink**.

Note When creating an NPC, you must specify the CPE as the source device and the PE-POP as the destination device. If there are intermediate devices, such as PE-CLEs, the source and destination devices must follow the direction of the CPE to PE-POP link.

Table 8-11 Create Named Physical Circuits

Operation	className	Required Parameters
createInstance	NamedPhysicalCircut	PhysicalLink
	PhysicalLink	• SrcDevice • DestDevice • SrcIfName • DestIfName

You can create one XML request for the **NamedPhysicalCircuit** and include multiple **PhysicalLinks** as shown in the following example:

```
<ns1:createInstance>
     <objectPath xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">NamedPhysicalCircuit</className>
      <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
      </properties>
      <objectPath xsi:type="ns1:CIMObjectPath">
           <className xsi:type="xsd:string">PhysicalLink</className>
           <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">SrcDevice</name>
             <value xsi:type="xsd:string">Device1</value> </item>
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">DestDevice</name>
             <value xsi:type="xsd:string">Device2</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">SrcIfName</name>
              <value xsi:type="xsd:string">Intf1/0</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">DestIfName</name>
              <value xsi:type="xsd:string">Intf2/1</value> </item>
            </properties>
      <objectPath xsi:type="ns1:CIMObjectPath">
           <className xsi:type="xsd:string">PhysicalLink</className>
           <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">SrcDevice</name>
             <value xsi:type="xsd:string">Device3</value> </item>
             <item xsi:type="ns1:CIMProperty">
             <name xsi:type="xsd:string">DestDevice</name>
             <value xsi:type="xsd:string">Device5</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">SrcIfName</name>
              <value xsi:type="xsd:string">Intf3/0</value> </item>
             <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">DestIfName</name>
              <value xsi:type="xsd:string">Intf5/1</value> </item>
            </properties>
       </objectPath>
      </objectPath>
   </ns1:createInstance>
```

XML Examples:

- CreateNamedPhysicalCircuit.xml
- CreateNamedPhysicalCircuitRing.xml—Use this example if there is a Ring topology configuration on the PE-CLEs.
- CreateNamedPhysicalCircuitRingExisting.xml—Use this example to reference an NPC ring that has already been created.

Step 12 Create VLAN ID pool.

Create a VLAN ID pool, specify a range, and associate it to an access domain to manually enter the parameters for a VLAN ID pool. To have ISC automatically assign VLANs to the attachment circuits, specify the **Autopick_Vlan_ID** keyword in the service definition (Step 15).

When provisioning for an Ethernet core, VPLS service requests use the VLAN ID to reference the attachment circuits.

Table 8-12 Create VLAN ID Pools

Operation	className	Required Parameters
createInstance	VlanIdPool	• Start • Size • AssocClassType • AssocClassId

XML Examples:

- CreateVlanIdPool.xml

Step 13 Create VC ID pool.

For a VPLS VPN, all PE-POP routers use the same VC ID to establish the virtual circuit (VC) across the provider core. The VC ID is also the VPN ID and is assigned from the VC ID pool. ISC ensures that VC IDs are unique among VPLS VPNs.

Note A VC ID pool is global (not associated with a provider or organization).

Table 8-13 Create VC ID Pool

Operation	className	Required Parameters
createInstance	VcIdPool	• Start • Size

XML Examples:

- CreateVcIdPool.xml

Step 14 Create a VPN.

When you create a VPN to use in VPLS provisioning, you must enable it to support VPLS (**VplsVpn=true**), and define the type of service (**ERS** or **EWS**).

ISC assigns a VPN ID (from the VC ID pool) to each VPLS VPN.

Table 8-14 Create VPNs

Operation	className	Required Parameters
createInstance	VPN	• Name • Organization or Provider • VplsVpn=true • ServiceType= – ERS – EWS

XML Examples:

- CreateVPN.xml

Step 15 Create the VPLS service definition (policy).

A VPLS service definition specifies the core type, service subtype, device properties, and the attributes common to all attachment circuits.

Table 8-15 Create a VPLS Service Definition

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=Vpls • Provider or Organization **Note** If you do not specify a **Provider** or **Organization**, the service policy is global • ServiceDefinitionDetails
	ServiceDefinitionDetails	• SubType= – VPLS_EWS – VPLS_EWS_NO_CE – VPLS_ERS – VPLS_ERS_NO_CE • Core_Type= – MPLS – Ethernet • PE_Encap and CE_Encap= – DOT1Q – DEFAULT **Note** If you specify **DEFAULT**, define the port type with the **Use_Native_Vlan** attribute. – **Use_Native_Vlan=true** for Trunk with Native VLAN. – **Use_Native_Vlan=false** for Access Port. • VplsUniMacAddress – MacAddress (You can list multiple secure MAC addresses) • Autopick_Vlan_ID

Note If **Autopick_Vlan_ID=true**, be sure that an access domain is attached to the PE-POP, and a VLAN ID pool is assigned to the access domain (Step 7).

XML Examples:

- CreateVPLSServiceDefn_EWS.xml
- CreateVPLSServiceDefn_EWS_NO_CE.xml
- CreateVPLSServiceDefn_ERS.xml
- CreateVPLSServiceDefn_ERS_NO_CE.xml

Step 16 Create the VPLS service request.

An VPLS service request defines the service definition and VPN, assigns interfaces and attributes for each attachment circuit (**VplsLink**), and applies any template information.

Table 8-16 Create a VPLS Service Request

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • Provider or Organization **Note** If you do not specify a **Provider** or **Organization**, the service policy is global. • ServiceRequest
	ServiceRequest	• RequestName • Type=Vpls • ServiceRequestDetails

Table 8-16 Create a VPLS Service Request (continued)

Operation	className	Required Parameters
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=Vpls • VPN • VplsLink
	VplsLink	• NPC *or* • ManualConfig=true – PE – CE (not required for policy subtypes with no CE present) **Note** You can use either **NPC** or **ManualConfig** to define the VplsLInk interfaces. If you use **ManualConfig**, you must also specify the interfaces (**CE_Intf_Name**, **UNIDeviceInterface** for NO_CE subtypes, and **PE_Intf_Name**). • VplsUniMacAddress • Link Template (optional) **Note** See the "Templates in a Service Request" section on page 4-9.

Tip Record the **LocatorId** value from the XML response for the service request. The locator ID is required for subsequent service request tasks.

XML Example:

- CreateVPLSServiceOrder_ERS.xml
- CreateVPLSServiceOrder_ERS_NO_CE.xml
- CreateVPLSServiceOrder_EWS.xml
- CreateVPLSServiceOrder_EWS_NO_CE.xml

Auditing Service Requests

A configuration audit occurs automatically each time you deploy a service request. During this configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. An audit also verifies that there were no errors during deployment by examining the commands configured by the service request on the target devices. If the device configuration does not match what is defined in the service request, the audit flags a warning and sets the service request to a *Failed Audit* or *Lost* state.

If you do not want the configuration audit to occur, change the value for the **Audit** parameter. The **Audit** parameter supports these values:

- **Audit**—This is the default. A successfully deployed service request is automatically audited unless this flag is changed.
- **NoAudit**—Do not perform a configuration audit when the service request is deployed.
- **ForceAudit**—Perform a configuration audit even if the service request deployment is not successful.

You can use the Audit parameter with a **Create**, **Modify**, or **Decommission** service request or a **Deployment** task. See the "Service Decommission" section on page 3-10 for more information. To perform a configuration audit as a separate task, see the "Configuration Audit" section on page 3-11.

QoS Provisioning

Cisco IP Solution Center (ISC) supports quality of service (QoS) provisioning at the access circuit (CPE-to-PE link). ISC can provision QoS policies for a network independent of VPN services (IP QoS), or in addition to VPN services that have been provisioned by ISC.

Standard IP QoS is applied using service level and link level policies and is independent of VPN services. IP QoS provisioning is described in the "Provisioning Example for IP QoS" section on page 9-8.

QoS policies for VPN services are deployed on top of an existing VPN service request, such as MPLS VPN and L2VPN. ISC derives interface configuration information from the VPN service and applies the QoS policy to the interfaces. This type of QoS provisioning is described in the "Provisioning QoS for Existing Services" section on page 9-17.

This chapter describes QoS service concepts and the steps required to provision QoS services using the ISC API. The provisioning example includes the process flow from creating the inventory to auditing the service deployment.

For more information on QoS provisioning using ISC, refer to the *Cisco IP Solution Center Quality of Service User Guide, 4.1*.

This chapter contains the following sections:

QoS Service Definitions

A QoS service definition specifies the QoS policy, which includes the service class policy and the QoS link profile. Policy attributes and service class attributes are defined in the **IPQoS** service definition details. The link profile is defined in the **Link** service definition details (or **EoMplsLink** for an L2VPN service request).

The QoS service definition can be a **QoS**, **Link**, or **EoMplsLink** policy subtype.

- The **QoS** policy consists of policy attributes and service class attributes. The service class contains additional QoS policy attributes, traffic classification, and avoidance properties. Use **QoS** to define the QoS policy and service class policy for an IP QoS policy.

- The **Link** policy (also called the link profile) consists of bandwidth, aggregated traffic shapers, link efficiency, and interface-based aggregated rate limiters. Use **Link** to define the link profile for an IP QoS policy.
- The **EoMplsLink** is the link profile for L2VPN links, and consists of the committed information rate (**CirBps**) and the committed burst (**BurstBytes**). Use the **EoMplsLink** to define the link profile for a QoS service request created from an L2VPN service request.

QoS Policy

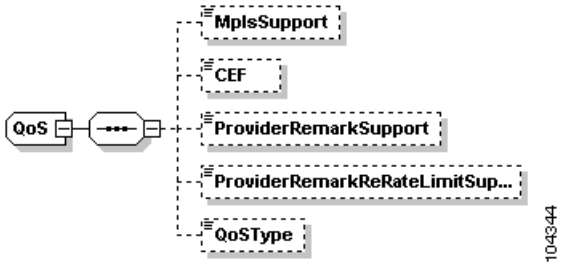
The QoS policy consists of policy attributes and service class attributes.

- Policy attributes include MPLS support for PE devices at the provider ingress interfaces. ISC supports remarking and rate-limiting at the provider ingress interface.
- Service class attributes include; traffic classification, congestion management, traffic shaping, rate limiting, and congestion avoidance.

QoS Attributes

Figure 9-1 shows the schema diagram for QoS policy attributes.

Figure 9-1 QoS Policy Schema Diagram



ISC supports the following properties for **QoS**:

- **MplsSupport**—Mark MPLS experimental (MPLS Exp.) value at provider ingress interface.
- **CEF**—Cisco Express Forwarding.
- **ProviderRemarkSupport**—Remark for differentiated services code point (DSCP) or IP Precedence value at provider ingress interface.
- **ProviderReRateLimitSupport**—Remark for rate limiting at provider ingress interface.
- **QoSType**—Specify IP QoS or Ethernet QoS.

Service Class Attributes

The service level QoS policy is the set of rules or conditions that apply to packets as they come across each device interface that has been assigned as a QoS link endpoint. This set of rules is defined in a QoS service class.

A QoS service class can include:

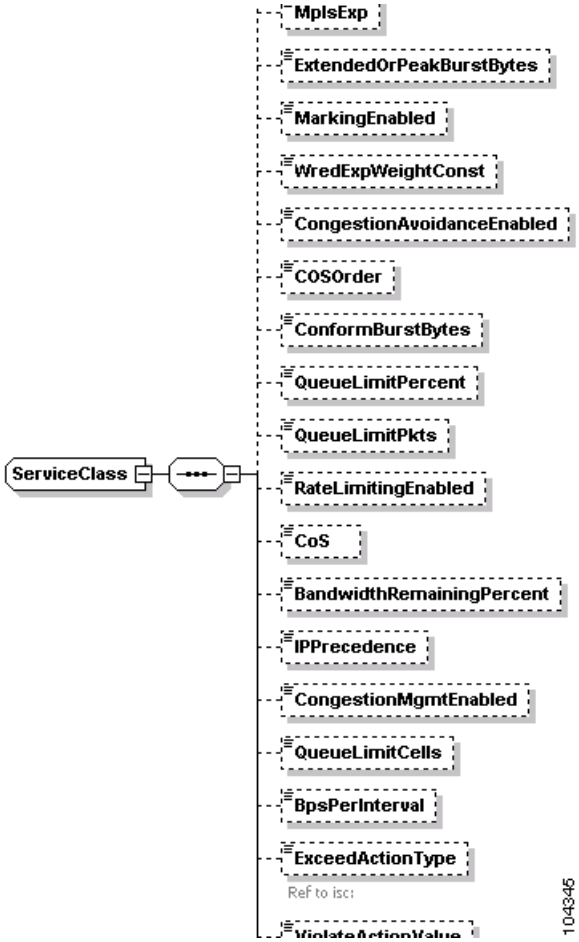
- Traffic classification (protocol, DSCP value, IP precedence value, source address)
- Traffic marking (DSCP or IP precedence values)
- Traffic shaping (average/peak, rate)
- Rate limiting (mean/peak rate, burst size, conform/exceed/violate actions)
- Congestion management (bandwidth and queue limit)
- Congestion avoidance (drop, exponential weighing constant)

Most networks require at least one service class for interactive voice traffic, one for management traffic, and at least one service class for data traffic. ISC supports the following service class subtypes:

- **VOIP**
- **DATA**
- **BEST_EFFORT** (data)
- **ROUTING_PROTOCOL**
- **MANAGEMENT**

The properties for each service class in the QoS policy are defined in the service definition details. Figure 9-2 shows a partial schema diagram for a QoS policy service class.

Figure 9-2 Service Class Schema Diagram



The following XML example shows a partial list of the properties that can be specified for a service class with **Type=DATA**:

Tip To *unset* integer type attributes in an XML request, set the value to -1.

```
<objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">ServiceClass</className>
            <properties xsi:type="ns1:CIMPropertyList"
                        soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Name</name>
                <value xsi:type="xsd:string">Service1A</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">Type</name>
                <value xsi:type="xsd:string">DATA</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">ConformAction</name>
                <value xsi:type="xsd:string">transmit</value>
                <qualifier xsi:type="ns1:CIMQualifier">
```

```
        <name xsi:type="xsd:string">value</name>
        <value xsi:type="xsd:string">2</value>
      </qualifier>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">ExceedAction</name>
      <value xsi:type="xsd:string">drop</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">value</name>
        <value xsi:type="xsd:string">1</value>
      </qualifier>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">ViolateAction</name>
      <value xsi:type="xsd:string">set-prec-transmit</value>
      <qualifier xsi:type="ns1:CIMQualifier">
        <name xsi:type="xsd:string">value</name>
        <value xsi:type="xsd:string">3</value>
      </qualifier>
    </item>
  <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">MarkingEnabled</name>
      <value xsi:type="xsd:string">TRUE</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">DSCP</name>
      <value xsi:type="xsd:string">ef</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">ShapingEnabled</name>
      <value xsi:type="xsd:string">TRUE</value>
     </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">BpsShapingRate</name>
      <value xsi:type="xsd:string">100000000</value>
    </item>
  <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">CongestionMgmtEnabled</name>
      <value xsi:type="xsd:string">TRUE</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">BandwidthPercent</name>
      <value xsi:type="xsd:string">15</value>
    </item>
```

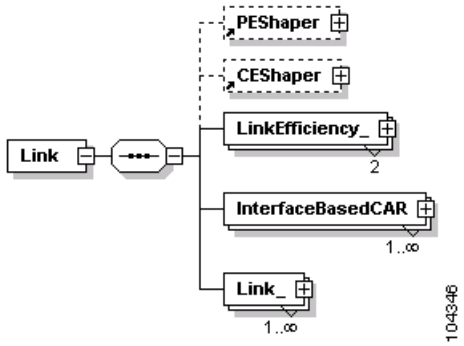
Link Policy

The **Link** portion of a QoS service definition (called the link profile) defines policy attributes specific to the CPE-to-PE links.

- Use **Link** for an IP QoS link profile. **Link** attributes are described in this section.
- Use **EoMplsLink** for an L2VPN link profile. **EoMplsLink** is described in the "L2VPN Ethernet QoS Policy" section on page 9-18.

Figure 9-3 shows the schema diagram for an IP QoS link profile.

Figure 9-3 QoS Link Profile Schema Diagram



IP QoS link profile attributes include:

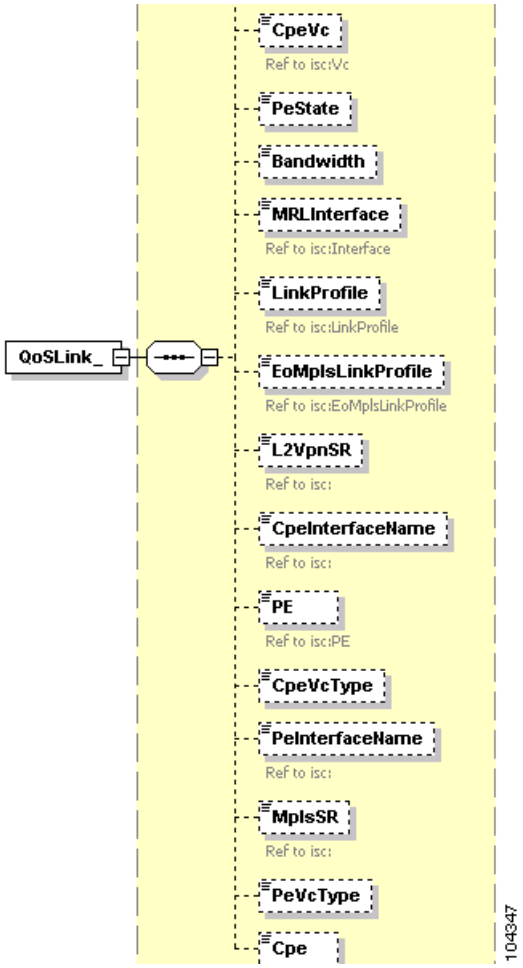
- Aggregated traffic shaper types on the CE and PE devices, such as; Frame-Relay traffic shapers, ATM traffic shapers, and parent level traffic shapers for nested policies. (**PEShaper**, **CEShaper**).
- Link efficiency settings such as; FRF.12, LFI on MLPPP, and cRTP (**LinkEfficiency**).
- Interface-based aggregated rate limiters such as; traffic classification, direction, mean rate, burst size, and conform/exceed actions (**InterfaceBasedCAR**).
- Link bandwidth (**Link**).

Note The link bandwidth specified in the **Link** profile overrides any link bandwidth specified in the **QoSLink** parameter of the service request.

QoS Service Requests

A QoS service request contains the service definition, which includes the QoS link profile and the QoS policy, and the **QoSLink** properties. The **QoSLink** properties specify the link endpoint devices, the link bandwidth, and link policy attributes (see Figure 9-4).

Figure 9-4 QoS Service Request Schema Diagram



The following properties are specified with the **QoSLink** object:

- The PE and CPE devices and interfaces for the link endpoints
  - **PE**, **PeInterfaceName**, **PeVcType**
  - **Cpe**, **CpeInterfaceName**, **CpeVcType**
- The link profile to assign to this QoS link
  - **ServiceDefinitionType=Link**
  - **ServiceDefinition=EoMplsLinkProfile**

ISC supports three methods for creating QoS service requests:

- IP QoS service request, independent of VPN services
- QoS service request created from an MPLS VPN service request
- QoS service request created from an L2VPN service request

IP QoS is described in the following provisioning example. Refer to the "Provisioning QoS for Existing Services" section on page 9-17 for information on QoS for VPN services.

Provisioning Example for IP QoS

This section describes the required steps for using the API to provision IP QoS and includes the operation, className, and required parameters for each step.

Prequisites

Note If you plan to use a metro Ethernet QoS policy (**QoSType**= **METROQOS**) on a Catalyst 3550 device, you must enable QoS on the device by entering the **mls qos** command. ISC does not provision this command on the device automatically, but it is required for the device to accept QoS configuration.

Process Summary

This QoS provisioning example includes the following steps:

- Create inventory
- Collect device configurations
- Mark device interfaces for QoS
- Create a network object for traffic classification (optional)
- Create a QoS service definition (policy)
- Create the IP QoS link profile (second part of the service definition)
- Create the IP QoS service request

Provisioning Process

This section provides an example provisioning process using QoS XML examples. The inventory of XML examples for the ISC API can be found at the following location:
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/isc/4_1/api_set/api_ref/examples/index.htm

Create Inventory

Every network element that ISC manages must be defined as a device in the system. An element is any device from which ISC can collect configuration information.

Step 1 Create devices.

In most cases, devices are Cisco IOS routers and Catalyst switches.

Table 9-1 Create Devices

Operation	className	Required Keywords
createInstance	• CiscoRouter • CatOs	One or more of the following: • ManagementIPAddress • HostName • DomainName

XML Example:

- CreateCiscoRouter.xml

Step 2 Create a Provider and Region.

The provider is the administrative domain of an Internet service provider (ISP), with one border gateway protocol (BGP) autonomous system (AS) number. The network owned by the provider is called the backbone network. If an ISP has two AS numbers, you must define it as two provider administrative domains. Each provider can contain multiple regions.

Table 9-2 Create Provider and Region

Operation	className	Required Keywords
createInstance	Provider	• Name • AsNumber
	Region	• Name • Provider

XML Examples:

- CreateProvider.xml
- CreateRegion.xml

Step 3 Declare devices as PEs and assign them to regions.

Table 9-3 Create PEs

Operation	className	Required Keywords
createInstance	PE	• Provider • Region • Role= – PE_CLE – PE_POP • Device • Interface

XML Example:

- CreatePE.xml

Step 4 Create a customer (Organization) and site.

Table 9-4 Create Customer and Site

Operation	className	Required Keywords
createInstance	Organization	• Name
	Site	• Name • Organization

XML Examples:

- CreateOrganization.xml
- CreateSite.xml

Step 5 Declare devices as CPEs and assign them to sites.

When you declare a device as a CPE, you also specify a **ManagementType** and interface information.

Table 9-5 Create CPEs

Operation	className	Required Keywords
createInstance	Cpe	• Site • Device • ManagementType

XML Example:

- CreateCpe.xml

Collect Device Configurations

A device configuration collection is a task. This task uploads the current configuration from the device to the ISC database. The collection task executed through a service request, and the service request is scheduled through a service order.

Table 9-6 Collect Device Configurations

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberofRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=Task • ServiceRequestDetails
	ServiceRequestDetails	• SubType=Collection • Device (or DeviceGroup) **Note** You must select at least one device or device group. • RetrieveVersion=true • RetrieveDeviceInterfaces=true

The following example is a partial XML request used to collect the configuration from device enqospe1.

```
<objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">ServiceRequestDetails</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
           <name xsi:type="xsd:string">SubType</name>
           <value xsi:type="xsd:string">COLLECTION</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Device</name>
          <value xsi:type="xsd:string">enqospe1</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">RetrieveDeviceAttributes</name>
          <value xsi:type="xsd:string">true</value> </item>
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">RetrieveDeviceInterfaces</name>
          <value xsi:type="xsd:string">true</value> </item>
       </properties>
       </objectPath>
```

XML Example:

- CreateTaskServiceOrderCollection.xml

Mark Device Interfaces for QoS

For QoS provisioning, you must mark both interfaces in the CPE-to-PE link. Typically, the service provider supplies the list of devices and interfaces to be marked for QoS provisioning. CPE devices are generally marked at provider-facing interfaces, and PE devices at customer-facing interfaces.

Create an XML request to modify the device, and mark the interface as a QoS link endpoint.

The following example shows a **modifyInstance** operation for CPE device enqosce52 and specifies **QoSLinkEndpoint=true** for interfaceATM1/0.52.

```
<ns1:modifyInstance>
      <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
        <className xsi:type="xsd:string">Cpe</className>
        <keyProperties xsi:type="ns1:CIMKeyPropertyList"
            soapenc:arrayType="ns1:CIMKeyProperty[]">
          <item xsi:type="ns1:CIMKeyProperty">
            <name xsi:type="xsd:string">Device</name>
            <value xsi:type="xsd:string">enqosce52</value>
          </item>
        </keyProperties>
        <objectPath subAction="createInstance" xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">Interface</className>
          <properties xsi:type="ns1:CIMPropertyList"
              soapenc:arrayType="ns1:CIMProperty[]">
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">Name</name>
              <value xsi:type="xsd:string">ATM1/0.52</value>
            </item>
              <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">QoSLinkEndpoint</name>
              <value xsi:type="xsd:string">true</value>
            </item>
          </properties>
        </objectPath>
      </objectPath>
    </ns1:modifyInstance>
```

Table 9-7 Mark Interfaces

Operation	className	Required Parameters
modifyInstance	PE (or Cpe)	• Device • Interface
	Interface	• Name • QoSLinkEndpoint=true • QoSMarkingRL=true **Note** Use **QosLinkEndpoint** to mark provider-facing PE and CE interfaces and **QoSMarkingRL** to mark customer-facing CE interfaces.

XML Examples:

- ModifyPE.xml
- ModifyCpe.xml

Creating a Network Object for Traffic Classification

This step is optional.

Create a network object to classify traffic based on network addresses contained in variables, instead of based on protocols. The **Name** value of the network object is used later in the process, in the traffic classification of a **ServiceClass** when you define the QoS policy in the service definition.

Table 9-8 Create Network Object

Operation	className	Required Keywords
createInstance	NetworkObject	• Name • Cpe • Type=NETWORK • Value

See the following example:

```
<ns1:createInstance>
    <objectPath xsi:type="ns1:CIMObjectPath">
     <className xsi:type="xsd:string">NetworkObject</className>
     <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
     <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">Name</name>
      <value xsi:type="xsd:string">address_1</value>
      </item>
     <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">Cpe</name>
      <value xsi:type="xsd:string">enqosce52</value>
     </item>
     <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">Type</name>
      <value xsi:type="xsd:string">NETWORK</value>
     </item>
     <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">Value</name>
      <value xsi:type="xsd:string">12.12.12.0/24</value>
     </item>
     </properties>
    </objectPath>
   </ns1:createInstance>
```

XML Example:

- CreateNetworkObject.xml

Creating a QoS Policy

A QoS policy is a template of the parameters needed to define a QoS service request. After you define the policy template, it can be used by all QoS service requests that share a common set of attributes.

The QoS Policy consists of:

- Policy attributes—support for PE devices in an MPLS VPN network and re-marking and rate-limiting at the provider ingress interface.

- Service class attributes—traffic classification, congestion management, traffic shaping, rate limiting, and congestion avoidance.

This section describes how to set the **QoS** attributes of an IP QoS service definition. See the "Creating the IP QoS Link Profile" section on page 9-15 for information on setting the **Link** attributes.

Table 9-9 Create QoS Policy

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=QoS • ServiceDefinitionDetails
	ServiceDefinitionDetails	• QoSType= – IPQOS – METROQOS • ServiceClass • TrafficClassification • Provider or Organization **Note** If you do not specify a **Provider** or **Organization**, the policy is global.
	ServiceClass	• Type= – VOIP – DATA – BEST_EFFORT – ROUTING_PROTOCOL – MANAGEMENT • TrafficClassification • Avoidance **Note** (You can have multiple traffic classification and avoidance parameters.)

To use a variable for traffic classification, include the name of a Network Object variable that has already been created.

In the following XML example, the **address_1** variable is called in the **TrafficClassification** class. This variable was created in the "Creating a Network Object for Traffic Classification" section on page 9-13.

```
<objectPath xsi:type="ns1:CIMObjectPath">
        <className xsi:type="xsd:string">TrafficClassification</className>
        <properties xsi:type="ns1:CIMPropertyList"
           soapenc:arrayType="ns1:CIMProperty[]">
         <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">FromAddressVariable</name>
            <value xsi:type="xsd:string">address_1</value>
          </item>
        </properties>
```

```
</objectPath>
```

XML Example:

- CreateServiceDefnPolicy.xml

Creating the IP QoS Link Profile

This section describes how to set the **Link** attributes of an IP QoS service definition. See the "Creating a QoS Policy" section on page 9-13 for information on setting the **QoS** attributes.

Note For L2VPN, use an EoMPLS link profile. See the "L2VPN Ethernet QoS Policy" section on page 9-18 for more information.

The IP QoS link profile sets the QoS **Link** attributes. The **Link** attributes include:

- Link bandwidth
- Aggregated traffic shapers
- Link efficiency settings
- Interface-based aggregated rate limiters

Table 9-10 Create Link Profile

Operation	className	Required Parameters
createInstance	ServiceDefinition	• Name • Type=Link • ServiceDefinitionDetails – Bandwidth
	ServiceDefinitionDetails	• PEShaper • CEShaper • LinkEfficiency • InterfaceBasedCAR

XML Example:

- CreateServiceDefnLink.xml

Creating an IP QoS Service Request

This section describes how to create a service request for IP QoS. The QoS service request is deployed through a service order. For information on provisioning QoS for an existing L2VPN or MPLS VPN service request, see the "Provisioning QoS for Existing Services" section on page 9-17.

A QoS service request contains the service definition, which includes the QoS link profile and the QoS policy, and the **QoSLink** properties. The **QoSLink** properties specify the link endpoint devices, link bandwidth, and link policy attributes.

Note Use **performBatchOperation** to group the service order and service request into one XML request.

Table 9-11 Create Service Request

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=QoS • ServiceRequestDetails
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=QoS • QoSLink – Bandwidth **Note** The bandwidth specified in the **Link** profile overrides any link bandwidth specified in the **QoSLink** parameter of the service request.
	QoSLink	• PE – PeInterfaceName – PeVcType • Cpe – CpeInterfaceName – CpeVcType **Note** Specify the PE interface information (**PE**, **PeInterface**, and **PeVcType)** the CPE interface information (**Cpe**, **CpeInterfaceType**, and **CpeVcType)**, or both. • ServiceDefinition – ServiceDefinitionType=Link

Tip Record the **LocatorId** value that is returned in the XML response for the service request. This value is required to perform a configuration audit of the service request.

XML Example:

- CreateQoSServiceOrder.xml

Auditing Service Requests

A configuration audit occurs automatically each time you deploy a service request. During this configuration audit, ISC verifies that all Cisco IOS commands are present and that they have the correct syntax. An audit also verifies that there were no errors during deployment by examining the commands configured by the service request on the target devices. If the device configuration does not match what is defined in the service request, the audit flags a warning and sets the service request to a *Failed Audit* or *Lost* state.

If you do not want the configuration audit to occur, change the value for the **Audit** parameter. The **Audit** parameter supports these values:

- **Audit**—This is the default. A successfully deployed service request is automatically audited unless this flag is changed.
- **NoAudit**—Do not perform a configuration audit when the service request is deployed.
- **ForceAudit**—Perform a configuration audit even if the service request deployment is not successful.

You can use the Audit parameter with a **Create**, **Modify**, or **Decommission** service request or a **Deployment** task. See the "Service Decommission" section on page 3-10 for more information. To perform a configuration audit as a separate task, see the "Configuration Audit" section on page 3-11.

Provisioning QoS for Existing Services

If your VPN services have been provisioned by ISC, you can add QoS policies to an existing service request. The following QoS attributes can be added to existing VPN services:

- For an L2VPN or VPLS network—Rate limiting, configured at the UNI.

  Note QoS policies applied to L2VPN or VPLS networks must be Ethernet QoS policies (*not* IP QoS).

- For an MPLS VPN network—Marking packets with MPLS Exp. values, configured at the PE device ingress interface.

When you provision QoS for an existing service, the preliminary operations of creating inventory, collecting configurations, and marking device endpoints are not required. You provision QoS for VPN services that are already in the *deployed* state, and the preliminary operations are specified within the existing L2VPN or MPLS VPN service request. During deployment, the QoS service request relies on port/interface configuration from the existing services.

QoS is provisioned with a service order that includes a QoS service definition, which defines the QoS policy and the appropriate link profile, and the L2VPN or MPLS VPN service request. The QoS service request is deployed through the service order.

L2VPN Ethernet QoS Policy

For an L2VPN network, ISC applies QoS rate-limiting parameters to traffic as it leaves the CLE device (the PE device UNI).

To provision QoS for an already deployed L2VPN service request, you need the L2VPN service request locator ID and a service definition with an EoMPLS link profile (**Type=EoMplsLink**). The EoMPLS link profile includes the committed information rate (**CirBps**) and the committed burst (**BurstBytes**).

Table 9-12 L2VPN Ethernet QoS Policy

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=QoS • ServiceRequestDetails
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=QoS • QoSLink – Bandwidth
	QoSLink	• L2VpnSR – Name – LocatorId

XML Example:

- CreateQoSServiceOrderEoMPLS.xml

Provisioning QoS from an MPLS VPN Service Request

For an MPLS VPN network, ISC marks packets with an MPLS Exp. value at the PE device ingress interface.

ISC supports the following QoS parameters for MPLS VPNs:

- IP QoS, based on DSCP or IP Precedence values, before the packet enters the MPLS VPN network.
- Map DSCP or IP Precedence value to MPLS Exp. value at the ingress router to the MPLS VPN Network (PE device ingress interface).
- Per-hop-behaviors (PHBs), such as congestion management and congestion avoidance in the MPLS backbone based on the MPLS Exp. value.
- IP QoS, based on DSCP or IP Precedence values, continues after the packet leaves the MPLS VPN network.

Table 9-13 MPLS QoS Policy

Operation	className	Required Parameters
createInstance	ServiceOrder	• ServiceName • NumberOfRequests • ServiceRequest
	ServiceRequest	• RequestName • Type=QoS • ServiceRequestDetails
	ServiceRequestDetails	• ServiceDefinition – ServiceDefinitionType=QoS • QoSLink – Bandwidth
	QoSLink	• MplsSR – Name – LocatorId **Note** A link profile is not required when provisioning QoS with an MPLS service request.

XML Example:

- CreateQoSServiceOrder.xml

GUI to API Mapping

This appendix maps the GUI operations to the corresponding APIs.

Service Inventory Tab

Table A-1 Service Inventory Tab

GUI Operation	Corresponding APIs
Inventory and Connection Manager	See the "Inventory and Connection Manager" section on page A-2.
Deployment Flow Manager	• performBatchOperations • Create<service type>ServiceOrder (where <service type> =MPLS, QoS, VPLS, L2VPN)
Device Console	• Download Commands (see also Template Manager on the Service Design Tab) – CreateTemplateServiceOrderDownload – CreateTemplateServiceOrderDownloadTransient • Download Template (see also Template Manager on the Service Design Tab) – CreateTemplateData, ModifyTemplateData – CreateTemplateDefn Delete and View also supported. • Device Configuration Manager – ViewConfiglet – ViewConfiguration • Exec Commands – CreateExecCommandServiceOrderDownload • Reload – CreateConfigServiceOrderDownload

Inventory and Connection Manager

Table A-2 Inventory and Connection Manager Link

GUI Operation	Corresponding APIs
Service Requests	Create<service type>ServiceOrder (where <service type> =MPLS, QoS, L2VPN, VPLS) Delete, Modify, and View also supported. (see also Task Manager on the Monitoring Tab)
Traffic Engineering Management	Not supported.
Inventory Manager	See individual inventory items.
Topology Tool	GUI only.
Devices	• CreateCiscoRouter • CreateCatIOS • CreateCatOS – CreateVPNSM • CreateIE2100 • CreateTerminalServer Delete, Modify, and View also supported.
Device Groups	CreateDeviceGroup Delete, Modify, and View also supported.
Customers	CreateOrganization Delete, Modify, and View also supported.
Providers	CreateProvider Delete, Modify, and View also supported.
Resource Pools	• CreateRouteTarget • CreateRouteDistinguisher • CreateIPAddressPool • CreateMulticastAddrPool • CreateSiteOfOrigin • CreateVcIdPool • CreateVlanIdPool Delete and View also supported.
CE Routing Communities	• CreateCERC • CreateVPN_DefaultCerc Delete, Modify, and View also supported.
VPNs	CreateVPN Delete, Modify, and View also supported.

Table A-2 Inventory and Connection Manager Link (continued)

GUI Operation	Corresponding APIs
AAA Servers	• CreateAAAServer • CreateAAAServerNTDomain • CreateAAAServerRADIUS • CreateAAAServerSDI • CreateAAAServerTACACS Delete, Modify, and View also supported.
Named Physical Circuits	• CreateNamedPhysicalCircuit • CreateNamedPhysicalCircuitRing • CreateNamedPhysicalCircuitRingExisting Delete, Modify, and View also supported.

Service Design Tab

Table A-3 Service Design Tab

GUI Operation	Corresponding APIs
Policies	Create<service type>ServiceDefn (where <service type> =MPLS, QoS, L2VPN) Delete, Modify, and View also supported.
Templates	• CreateTemplateData, ModifyTemplateData • CreateTemplateDefn • CreateTemplateServiceOrderDownload • CreateTemplateServiceOrderDownloadTransient Delete and View also supported. **Note** Not supported for QoS SR.
Protocols	Not supported.
Link QoS	CreateQoSServiceDefnLink Delete, Modify, and View also supported.
Network Objects	CreateNetworkObject Delete, Modify, and View also supported.

Monitoring Tab

Table A-4 Monitoring Tab

GUI Operation	Corresponding APIs
Task Manager	• CreateTaskServiceOrderConfigAudit • CreateTaskServiceOrderCollection • CreateTaskServiceOrderDecommission • CreateTaskServiceOrderDeployment • CreateTaskServiceOrderMplsFuncAudit Delete and View also supported.
Ping	CreateTaskServiceOrderMplsFuncAudit.xml (Ping is part of an MPLS functional audit.)
SLA	For SLA Probes: • CreateSLAProbe, CreateSLAProbeServiceRequest Delete, Modify and View also supported. For SLA Reports: • execReport_SLAHttp, exexReport_SLAHttpCos • execReport_SLAJitter, execReport_SLAJitterCos • execReport_SLASummary, exectReport_SLASummaryCos
TE Performance Report	Not supported.

Administration Tab

Table A-5 Administration Tab

GUI Operation	Corresponding APIs
Security	GUI only.
Control Center	• Hosts—Not supported. • Collection Zones – CreateCollectionZone Delete, Modify, and View also supported • Licensing—N/A
Active Users	N/A
User Access Log	N/A
Manage TIBCO Rendezvous	N/A

Implementing a Notification Server

This appendix describes how to modify the example servlet to customize for your own application, and contains the following sections:

Event Notification Overview

Event notification allows an ISC API user to collect events of interest from the ISC system. The events of interest are triggered by changes to objects managed by ISC. An event is registered and a notification is sent any time a database object is created, modified, or deleted, when a scheduled task begins or ends its execution, or when a watchdog event signals a change in execution status for any ISC service.

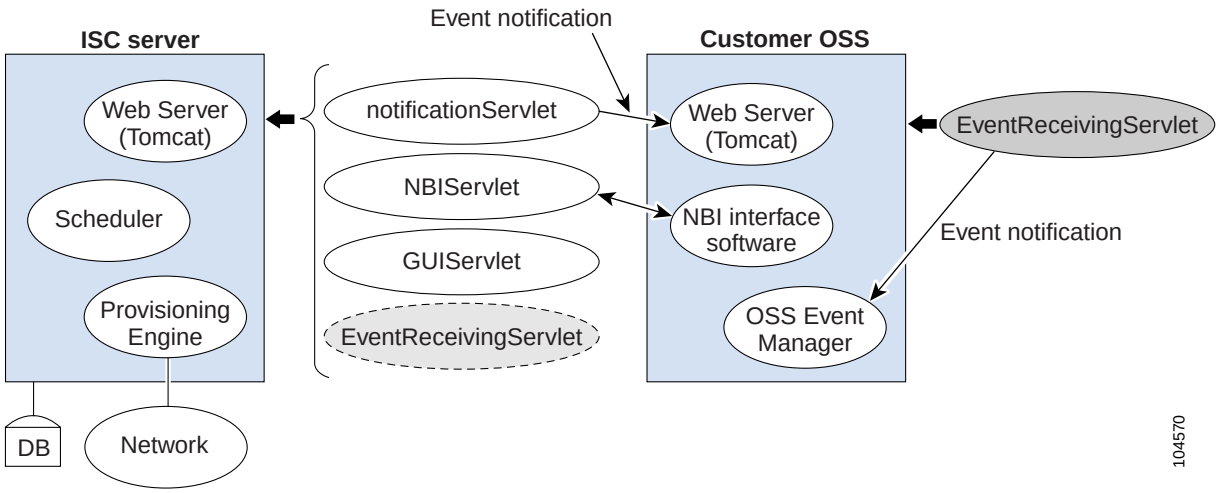
The ISC properties that manage the notification server are:

- notification.clientEnabled—Used to turn notification forwarding on and off.
- notification.clientHost—The machine running the event notification receiving program.
- notification.clientPort—The listener port to open on the receiving machine. This property is defined in the Tomcat web.xml file.
- notification.clientMethod—The method, or program, to contact on the receiving machine.
- notification.clientRegFile—Contains the events of interest to be forwarded.

Note clientHost, clientMethod, and clientPort are used to construct a URL.

Figure B-1 illustrates the notification mechanisms implemented by ISC.

Figure B-1 Event Notification Mechanisms



The **EventReceivingServlet** can be customized and used for your own application.

The notification web.xml file (located in $ISC_HOME/resources/webserver/tomcat/webapps/notification/WEB-INF) in ISC identifies two servlets:

- notificationServle
- eventListener (EventReceivingServlet program)

Use this file as a template for implementing a notification servlet.

In the following web.xml file, the **notificationServlet** section and **eventListener** section are indicated in **bold**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>notificationServlet</servlet-name>
    <display-name>Notification Servlet</display-name>
    <servlet-class>com.cisco.vpnsc.nbi.notification.NotificationServlet</servlet-class>
    <init-param>
      <param-name>log</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>eventListener</servlet-name>
    <display-name>Event Listener</display-name>
    <servlet-class>com.cisco.vpnsc.nbi.client.EventReceivingServlet</servlet-class>
    <init-param>
      <param-name>log</param-name>
      <param-value>2</param-value>
    </init-param>
```

```
    <load-on-startup>2</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>eventListener</servlet-name>
    <url-pattern>/notification/servlet/eventListener</url-pattern>
  </servlet-mapping>
</web-app>
```

The ISC installation contains an example servlet which can be activated to demonstrate the collection of notification events. The event notification receiving servlet (eventListener) calls a program com.cisco.vpnsc.nbi.client.EventReceivingServlet. This program is included in the ISC installation ($ISC_HOME/resources/nbi/client/EventReceivingServlet.java) and shows a simple piece of code which receives an event and prints it to a log file.

Note For more information on example clients, see Appendix C, "Scripts".

Running the Example Servlet

ISC includes an example event-notification client servlet. This servlet can be activated on a standard ISC installation.

The example effectively creates a loopback scenerario whereby an ISC event generates a SOAP/XML message that is sent back to the ISC server's HTTP port. Upon arriving at the port (8030 is the default), the Tomcat server loads and executes the EventReceivingServlet client code.

You can observe the outbound event message and the inbound notification message by inspecting $ISC_HOME/tmp/httpd_out.0 and $ISC_HOME/notification.0 respectively. If the event notification succeeds, you will see the SOAP/XML event notification at the bottom of the notification.0 file.

To run a demonstration of the example servlet:

Step 1 Add an event to the cilentReg.txt file. This file is located $ISC_HOME/resources/nbi/notification/clientReg.txt

Step 2 Enable dynamic loading of servlets on the ISC server with the following actions:

A) edit $ISC_HOME/resources/webserver/tomcat/conf/web.xml

B) Make sure the mapping for the invoker servlet-mapping is uncommented. The following is the XML text for the invoker servlet-mapping:

```
<!-- The mapping for the invoker servlet -->
  <servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
```

For this example, add **com.cisco.vpnsc.repository.devices.CiscoRouter.>** to specify events involving a CiscoRouter.

Step 3 Change the property **notification.clientEnabled** to true. This file is located in $ISC_HOME/etc/vpnsc.properties.

From the GUI:

Go to **Administration->Control Center**.

Choose your host and click **Config**.

Locate the notification server and select the **clientEnabled** property. Set to **true** and click **Set Property**.

Step 4 Restart ISC.

Step 5 Add a CiscoRouter device.

From the GUI:

Go to **Service Inventory** > **Inventory and Connection Manager** > **Devices.**

Click **Create** and select **Cisco IOS Device**.

Enter a host name.

Click **Save.**

Step 6 Go to $ISC_HOME/tmp and cat the file httpd_out.0. The XML at the end should look like the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message />
  </soapenv:Header>
  <soapenv:Body>
    <ns1:deliverEvent>
      <returns xsi:type="ns1:CIMReturnList" soapenc:arrayType="ns1:CIMReturn[]">
        <record>
          <objectPath xsi:type="ns1:CIMObjectPath">
            <className xsi:type="xsd:string">InstIndication</className>
            <properties xsi:type="ns1:CIMPropertyList"
soapenc:arrayType="ns1:CIMProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">IndicationType</name>
                <value xsi:type="xsd:string">create</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">IndicationTime</name>
                <value xsi:type="xsd:string">2003-12-03 13:15:26</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">InstClassName</name>
                <value xsi:type="xsd:string">CiscoRouter</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">InstName</name>
                <value xsi:type="xsd:string">xyz</value>
              </item>
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">LocatorId</name>
                <value xsi:type="xsd:string">40</value>
              </item>
            </properties>
          </objectPath>
        </record>
      </returns>
    </ns1:deliverEvent>
  </soapenv:Body>
```

```
</soapenv:Envelope>
```

Customizing the Example Servlet

Use the following procedure to implement a custom notification receiving servlet

Step 1 Copy the web.xml file, located in $ISC_HOME/resources/webserver/tomcat/webapps/notification/WEB-INF, to your own Tomcat structure.

Step 2 Remove the notification servlet xml section. This section is specific to ISC.

Step 3 Change the eventListener program from **com.cisco.vpnsc.nbi.client.EventReceivingServlet** to your own program.

Step 4 Specify the events to receive by editing the contents of the file $ISC_HOME/resources/nbi/notification/clientReg.txt. The default contents of this file are:

```
com.cisco.vpnsc.repository.task.PersistentTask.>
com.cisco.vpnsc.repository.devices.PIX.>
```

This means that events involving **PIX** devices or **PersistentTask** will be forwarded to the EventListener servlet.

For example, add **com.cisco.vpnsc.repository.devices.CiscoRouter.>** to specify events involving a CiscoRouter.

See the "Events for Collection" section on page B-5 for a complete list of events that can be collected and forwarded.

Step 5 Change the property **notification.clientEnabled** to true. This file is located in $ISC_HOME/etc/vpnsc.properties.

From the GUI:

Go to **Administration** >**Control Center**.

Choose your host and click **Config**.

Locate the notification server and select the **clientEnabled** property. Set to **true** and click **Set Property**.

Step 6 Restart ISC.

Events for Collection

The following events can be collected for the ISC notifications server:

- 'com.cisco.vpnsc.repository.task.Argument'
- 'com.cisco.vpnsc.repository.task.Action'
- 'com.cisco.vpnsc.repository.task.ActionDependency'
- 'com.cisco.vpnsc.repository.task.ActionTarget'

- 'com.cisco.vpnsc.repository.task.PersistentTask'
- 'com.cisco.vpnsc.repository.task.RuntimeAction'
- 'com.cisco.vpnsc.repository.task.RuntimeTask'
- 'com.cisco.vpnsc.repository.task.ScheduledTask'
- 'com.cisco.vpnsc.repository.devices.Device'
- 'com.cisco.vpnsc.repository.devices.CiscoDevice'
- 'com.cisco.vpnsc.repository.devices.CiscoRouter'
- 'com.cisco.vpnsc.repository.devices.PIX'
- 'com.cisco.vpnsc.repository.devices.TerminalServer'
- 'com.cisco.vpnsc.repository.devices.CatOs'
- 'com.cisco.vpnsc.repository.devices.NonCiscoDevice'
- 'com.cisco.vpnsc.repository.devices.IE2100'
- 'com.cisco.vpnsc.repository.devices.GenericDeviceInterface'
- 'com.cisco.vpnsc.repository.devices.CiscoDeviceInterface'
- 'com.cisco.vpnsc.repository.devices.NonCiscoDeviceInterface'
- 'com.cisco.vpnsc.repository.devices.QOSDeviceInterface'
- 'com.cisco.vpnsc.repository.devices.PIXDeviceInterface'
- 'com.cisco.vpnsc.repository.devices.DeviceGroup'
- 'com.cisco.vpnsc.repository.devices.RepDev_DevGp_Mpng'
- 'com.cisco.vpnsc.repository.devices.CollectionZone'
- 'com.cisco.vpnsc.repository.devices.VPNSCHost'
- 'com.cisco.vpnsc.repository.devices.VPNSCHostProperties'
- 'com.cisco.vpnsc.repository.devices.VirtualCircuit'
- 'com.cisco.vpnsc.repository.devices.AtmVC'
- 'com.cisco.vpnsc.repository.device.FrameRelayVC'
- 'com.cisco.vpnsc.repository.devices.VlanVC'
- 'com.cisco.vpnsc.repository.devices.PEDevice'
- 'com.cisco.vpnsc.repository.devices.CPEDevice'
- 'com.cisco.vpnsc.repository.common.Customer'
- 'com.cisco.vpnsc.repository.common.Site'
- 'com.cisco.vpnsc.repository.common.LogicalDevice'
- 'com.cisco.vpnsc.repository.common.Cpe'
- 'com.cisco.vpnsc.repository.common.CustomerNetworkPrefix'
- 'com.cisco.vpnsc.repository.common.VPN'
- 'com.cisco.vpnsc.repository.common.Interface'
- 'com.cisco.vpnsc.repository.common.Vc'
- 'com.cisco.vpnsc.repository.common.PE'
- 'com.cisco.vpnsc.repository.common.Provider'

- 'com.cisco.vpnsc.repository.common.Region'
- 'com.cisco.vpnsc.repository.mpls.VRF'
- 'com.cisco.vpnsc.repository.common.GenericSR'
- 'com.cisco.vpnsc.repository.mlshare.AccessDomain'
- 'com.cisco.vpnsc.repository.common.AttributeValue'
- 'com.cisco.vpnsc.repository.common.ConfigletClob'
- 'com.cisco.vpnsc.repository.common.Configlet'
- 'com.cisco.vpnsc.repository.common.Policy'
- 'com.cisco.vpnsc.repository.common.SRAssociatedTemplate'
- 'com.cisco.vpnsc.repository.common.RepCableIpAddress'
- 'com.cisco.vpnsc.repository.common.RepSRReport'
- 'com.cisco.vpnsc.repository.common.SRHistoryReport'
- 'com.cisco.vpnsc.repository.common.StagedConfiglet'
- 'com.cisco.vpnsc.repository.common.StagedSR'
- 'com.cisco.vpnsc.repository.common.GenericSRToVpn'
- 'com.cisco.vpnsc.repository.mlshare.PhysicalLink'
- 'com.cisco.vpnsc.repository.mlshare.Ring'
- 'com.cisco.vpnsc.repository.mlshare.PhysicalLinkSeq'
- 'com.cisco.vpnsc.repository.mlshare.NamedPhysicalCircuit'
- 'com.cisco.vpnsc.repository.mlshare.MLSharePolicy'
- 'com.cisco.vpnsc.repository.mlshare.MlConnPolicy'
- 'com.cisco.vpnsc.repository.mlshare.LogicalLink'
- 'com.cisco.vpnsc.repository.mlshare.LogicalLinkSet'
- 'com.cisco.vpnsc.repository.mlshare.EndPoint'
- 'com.cisco.vpnsc.repository.mlshare.DevEndPoint'
- 'com.cisco.vpnsc.repository.mlshare.RepNotEditable'
- 'com.cisco.vpnsc.repository.mlshare.LinkTemplate'
- 'com.cisco.vpnsc.repository.mlshare.ReservedVlanPool'
- 'com.cisco.vpnsc.repository.mlshare.RingMembership'
- 'com.cisco.vpnsc.repository.mlshare.RingSeq'
- 'com.cisco.vpnsc.repository.mpls.MplsSR'
- 'com.cisco.vpnsc.repository.mpls.CERC'
- 'com.cisco.vpnsc.repository.mpls.CERCMembership'
- 'com.cisco.vpnsc.repository.mpls.VRFRole'
- 'com.cisco.vpnsc.repository.mpls.MplsVpnLink'
- 'com.cisco.vpnsc.repository.mpls.MplsLogicalLink'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyAttributeMap'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicy'

- 'com.cisco.vpnsc.repository.mpls.MplsPolicyInterface'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyRoutingProtocol'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyOSPFRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyBGPRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyEIGRPRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyIGRPRouting'
- 'com.cisco.vpnsc.repository.l2vpn.AC'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyISISRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyRIPRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyStaticRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyNoneRouting'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyRedistributedRoutingProtocol'
- 'com.cisco.vpnsc.repository.mpls.MplsPolicyCercMembership'
- 'com.cisco.vpnsc.repository.mpls.RepStaticRoutes'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrs'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsAttrMap'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsRedistributedRoutingProtocol'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsRoutingProtocol'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsOSPFRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsBGPRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsEIGRPRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsIGRPRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsISISRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsRIPRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsStaticRouting'
- 'com.cisco.vpnsc.repository.mpls.LinkAttrsNoneRouting'
- 'com.cisco.vpnsc.repository.mpls.RoutingProtocolAccessor'
- 'com.cisco.vpnsc.repository.mpls.LinkRoutingProtocolAccessor'
- 'com.cisco.vpnsc.repository.mpls.RepCableIpAddr'
- 'com.cisco.vpnsc.repository.mpls.RepOldTemplate'
- 'com.cisco.vpnsc.repository.mpls.MplsUniEtth'
- 'com.cisco.vpnsc.repository.mpls.LinkMulticastIPAddr'
- 'com.cisco.vpnsc.repository.l2vpn.L2VpnPolicyAttributeMap'
- 'com.cisco.vpnsc.repository.l2vpn.L2VpnPolicy'
- 'com.cisco.vpnsc.repository.l2vpn.L2VpnPolicyInterface'
- 'com.cisco.vpnsc.repository.l2vpn.EndToEndWireAttributeMap'
- 'com.cisco.vpnsc.repository.l2vpn.ACAttributeMap'
- 'com.cisco.vpnsc.repository.l2vpn.EndToEndWire'

- 'com.cisco.vpnsc.repository.l2vpn.EndToEndWireAttrs'
- 'com.cisco.vpnsc.repository.l2vpn.ACAttrs'
- 'com.cisco.vpnsc.repository.l2vpn.L2VpnSR'
- 'com.cisco.vpnsc.repository.l2vpn.L2VpnLogicalLink'
- 'com.cisco.vpnsc.repository.l2vpn.UniInterface'
- 'com.cisco.vpnsc.repository.l2vpn.UniProtocolTunnel'
- 'com.cisco.vpnsc.repository.l2vpn.UniMacAccessList'
- 'com.cisco.vpnsc.repository.qos.QoSPolicy'
- 'com.cisco.vpnsc.repository.l2vpn.UniSecureMacAddress'
- 'com.cisco.vpnsc.repository.protocol.Protocol'
- 'com.cisco.vpnsc.repository.protocol.ProtocolBundle'
- 'com.cisco.vpnsc.repository.protocol.RepPro_Bdl_Mpng'
- 'com.cisco.vpnsc.repository.qos.QoSSR'
- 'com.cisco.vpnsc.repository.qos.QoSLink'
- 'com.cisco.vpnsc.repository.qos.LinkProfile'
- 'com.cisco.vpnsc.repository.qos.EoMplsLinkProfile'
- 'com.cisco.vpnsc.repository.qos.FrShaper'
- 'com.cisco.vpnsc.repository.qos.CbShaper'
- 'com.cisco.vpnsc.repository.qos.AtmShaper'
- 'com.cisco.vpnsc.repository.qos.LinkEfficiency'
- 'com.cisco.vpnsc.nbi.vpnsc.PLHelper'
- 'com.cisco.vpnsc.repository.qos.InterfaceBasedCAR'
- 'com.cisco.vpnsc.repository.qos.ServiceClass'
- 'com.cisco.vpnsc.repository.qos.TrafficClassification'
- 'com.cisco.vpnsc.repository.qos.Avoidance'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.DataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.AgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RdAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RdAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RdAgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RtAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RtAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RtAgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.SooAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.SooAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.SooAgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RdNameSeed'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RtNameSeed'

- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.SooNameSeed'
- 'com.cisco.vpnsc.repository.resourcePool.externallyManaged.PoolType'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.IPDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RdDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.RtDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.SooDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VcIdDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VlanDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.IPAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.IPAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.IPAgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.MCASTDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.MCASTAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.MCASTAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.MCASTAgeDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VlanAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VlanAvailDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VcIdAllocDataBlock'
- 'com.cisco.vpnsc.repository.resourcePool.vpnscManaged.VcIdAvailDataBlock'
- 'com.cisco.vpnsc.repository.collection.Data'
- 'com.cisco.vpnsc.repository.collection.DevOwner'
- 'com.cisco.vpnsc.repository.sla.SAAProbe'
- 'com.cisco.vpnsc.repository.sla.SAADNSProbe'
- 'com.cisco.vpnsc.repository.sla.SAAEchoProbe'
- 'com.cisco.vpnsc.repository.sla.SAAHTTPProbe'
- 'com.cisco.vpnsc.repository.sla.SAAJitterProbe'
- 'com.cisco.vpnsc.repository.sla.SAATCPConnectProbe'
- 'com.cisco.vpnsc.repository.sla.SAAUDPEchoProbe'
- 'com.cisco.vpnsc.repository.sla.SAADHCPProbe'
- 'com.cisco.vpnsc.repository.sla.SAAFTPProbe'
- 'com.cisco.vpnsc.repository.vpls.MacAddress'
- 'com.cisco.vpnsc.repository.vpls.VplsServiceAttributeMap'
- 'com.cisco.vpnsc.repository.vpls.VplsPolicyInterface'
- 'com.cisco.vpnsc.repository.vpls.VplsUniProtocolTunnel'
- 'com.cisco.vpnsc.repository.vpls.VplsUniInterface'
- 'com.cisco.vpnsc.repository.vpls.VplsPolicy'
- 'com.cisco.vpnsc.repository.vpls.VplsLinkAttributeMap'
- 'com.cisco.vpnsc.repository.vpls.VplsServiceAttributes'

- 'com.cisco.vpnsc.repository.vpls.VplsLink'
- 'com.cisco.vpnsc.repository.vpls.VplsSR'
- 'com.cisco.vpnsc.repository.common.HubGroup'
- 'com.cisco.vpnsc.repository.vpls.VplsLogicalLink'
- 'com.cisco.vpnsc.repository.common.MultipointHub'
- 'com.cisco.vpnsc.repository.deploymentflow.Service'
- 'com.cisco.vpnsc.repository.deploymentflow.DfProcess'
- 'com.cisco.vpnsc.repository.deploymentflow.Step'
- 'com.cisco.vpnsc.repository.deploymentflow.StepArg'
- 'com.cisco.vpnsc.repository.deploymentflow.NotifyUserId'
- 'com.cisco.vpnsc.repository.deploymentflow.GlobalData'
- 'com.cisco.vpnsc.repository.deploymentflow.StepData'
- 'com.cisco.vpnsc.repository.deploymentflow.DFProfileEntry'
- 'com.cisco.vpnsc.repository.ual.UALog'
- 'com.cisco.vpnsc.repository.ual.UALPolicy'
- 'com.cisco.vpnsc.repository.ual.ObjectLogEntry'
- 'com.cisco.vpnsc.repository.license.SoftwareLicense'
- 'com.cisco.insmbu.templatemgr.repository.TemplateGroup'
- 'com.cisco.insmbu.templatemgr.repository.Template'
- 'com.cisco.insmbu.templatemgr.repository.DataFile'
- 'com.cisco.insmbu.templatemgr.repository.Keyword'
- 'com.cisco.insmbu.templatemgr.repository.TemplateAssociationOnRole'
- 'com.cisco.insmbu.templatemgr.repository.BusinessClassAssociation'
- 'com.cisco.vpnsc.repository.nbi.ServiceOrder'
- 'com.cisco.vpnsc.repository.nbi.SOSRAssociation'

Scripts

These UNIS shell scripts automate the input of XML requests to, and process the resulting output from, the Northbound Interface (NBI) Application Programmers Interface (API) of the Cisco IP Solution Center (ISC) network management application.

This appendix contains information about the following scripts:

- changeMaxRoutes
- changepasswd
- collectConfig
- deletece
- deletesr
- deployallsr
- deploysr
- downinterface
- getpe
- modifyce
- purgesrs
- removesr
- showsr
- srdump
- taskdump
- upinterface
- VrfPing

README File

The README file contains an example of a working script file and describes the required environment variables and parameters, and the location for optional files.

Scripts Main directory

This section describes the scripts in the main directory. See the "Script Subdirectories" section on page C-15 for more information about these optional files required by the UNIX shell scripts in the main scripts directory.

Note These scripts work with either the Sybase and Oracle database.

env

The **env** file contains all of the environment or UNIX shell variable definitions required by all of the UNIX shell scripts in the main scripts directory. All existing UNIX shell scripts in this directory reference the **env** file. Any new scripts created must also include a reference to this file.

changeMaxRoutes

This script changes the maximum allowed VPN routes for the service request links that belong to the specified VPN and Customer. It also downloads the **maxRoutes** value to the PE devices that belong to the service request links.

Command Syntax

changeMaxRoutes [-v vpnName] [-c customerName] -m maxroutes

Table C-1 changeMaxRoutes Command Options

Option	Description
-v vpnName	VPN name. Optional parameter.
-c customerName	Customer name. Optional parameter.
-m maxroutes	The maximum number of VPN routes allowed in the device configuration. Required parameter.

STDOUT

```
State Success
OutputString
ilpe3.cisco.com|V1:test_vpn|change
ilpe3.cisco.com|V1:test_vpn|change
ilpe2.cisco.com|V2:test_vpn-s|change
ilpe2.cisco.com|V3:newvpn|nochange: Reason- since could not set maxroutes in repository
```

Where State is either *Success* or *Failure.* The OutputString is:

```
<pename>|<vrf name>|<change or nochange: Reason- >
```

LOG

The log information is stored in <ISC log Location>/http.0.* in xml format.

The information stored depends on the log level. Log levels range from SEVERE to FINEST, and are set using Administration -> Control Center->Hosts->Configuration->Logging->Default->Loglevel.

changepasswd

This script causes the ISC application to change the password on a specified device.

Command Syntax

changepasswd -f <inputfilename> [-log <logfilename>] | changepasswd -help

STDOUT

0 for success, 1 for failure

File Name

None

Log Name

The default log name is $ISC_HOME/tmp/changepasswd.log.$$, where $$ is the process ID. An alternate log file name can be specified in the input parameters.

Log Output Example

```
-------------------------------------
opening the repository
input: 3550_6-1|NbiRegion|test|test1|test2
rpmname: 3550_6-1
regionname: NbiRegion
newusername: test
newpassword: test1
newenpassword: test2
After constructTibrvMsg Password ID:: 43835
RPM 3550_6-1 Success
********
input:
```

collectConfig

Use this script to collect the device configuration for a specified device (**rpmName**). The device configuration is stored in the directory $ISC_HOME/tmp in a file named after the device. You can list multiple device names (multiple **rpmName** parameters).

Command Syntax

collectConfig -r rpmName

STDOUT

0 for success, 1 for failure.

File Name

$ISC_HOME/tmp/device, where *device* is the name of the device (for example, 3550_6-1).

File Output Example

```
3550_6-1#term len 0
3550_6-1#show run
Building configuration...

Current configuration : 10676 bytes
!
version 12.1
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname 3550_6-1
!
enable secret 5 $1$xHqv$.pVjEARI1vXrJ7tK1S0qa1
!
errdisable recovery cause l2ptguard
errdisable recovery interval 5000
ip subnet-zero
!
...
```

Log Name

$ISC_HOME/tmp/collectConfig.log

Log Output Example

```
Mon Aug  2 14:13:36 PDT 2004:  collectConfig started
--------------------------------------------
collectConfig request created for device: 3550_6-1
--------------------------------------------

saving config for device 3550_6-1 in the directory /opt/vpnsc/iscadmin/tmp
```

deletece

Deletes all CE devices in the repository that have no service requests associated with them.

Command Syntax

deletece [-p pvc_id]

Optionally, you can specify a single CE device using the **pvc_id** value with the **-p** script option flag.

STDOUT

```
Deleting unused CEs
c1234
The number of CEs deleted: 1
The number of Sites deleted: 0
To view the log file, please see /tmp/deletecefile
```

File Name

None

Log Name

$ISC_HOME/tmp/deletecefile

Log Output Example

```
Deleting unused CEs
Start TIME = Mon Aug  2 15:24:36 PDT 2004
c1234
The number of CEs deleted: 1
The number of Sites deleted: 0
End TIME = Mon Aug  2 15:25:07 PDT 2004
```

deletesr

This script performs these actions:

- Decommissions the list of service requests from the ISC database. The service request is specified using the **SRJobId**.
- Produces an audit report for all specified service requests, returning the associated **JobId** and state for each one.
- Purges service requests in the *Closed* state.

The audit report can be disabled using the **-noaudit** option flag.

Command Syntax

deletesr SRJobId [SRJobId, ...]

STDOUT

```
113     CLOSED - purging
Purge complete
```

deployallsr

This script performs these actions:

- Finds all the MPLS service requests that are in the *Requested* state.
- Deploys the above listed MPLS service requests to the ISC-managed network.

Command Syntax

deployallsr [-outdir dir_name] [-log log_file_name]

STDOUT

None

File Name

None

Log Name

$ISC_HOME/tmp/deployallsr.log.1897_08_03_04_09_06_29

Where 1897 is the process ID, and the remaining numbers are the date and time. An alternate log file name can be specified in the input parameters.

Log Output Example

```
SRs to be deployed...
146
Task deployment state: Completed
DateTime,SRJobID,State
2004-08-03 10:07:03,146,CLOSED
```

deploysr

This script performs these actions:

- Deploys all service requests listed in the input parameters, *regardless* of the state.
- Produces an audit report for all specified service requests, returning the associated **JobId** and state for each one.

Command Syntax

deploysr SRJobId [SRJobId, ...]

The audit report can be disabled using the **-noaudit** option flag.

STDOUT

None, unless there is an error. The following is an example of an error output:

```
The SR with ID: 1 does not exist in the Database!
```

File Name

None

Log Name

None

downinterface

Use this script to turn off or shut down a given network interface (**interfaceName**) on a given device (**rpmName**). This script logs into the listed RPM device and inserts the **shutdown** IOS command on the specified interface.

Command Syntax

downinterface -rpm rpmName [-user userName] -pw userPassword -enableuser enableUserName -enablepw enablePassword -interface interfaceName [-log Log filename]

Table C-2 downinterface Command Options

Command Option	Description
-rpm	Host name (or IP address) of the RPM (PE device). Required parameter.
-user	Login username. This parameter is only required if both the username and password are required for login.
-pw	Login password. Required parameter.
-enableuser	Enable username. This parameter is only required if both the username and password are required to enter enable mode.
-enablepw	Enable password. Required parameter.
-interface	The complete interface name (for example, Switch1.1). Required parameter.
-log	Log filename. Optional parameter. If not specified, the file downinterface.log is created in the $ECSP_HOME/tmp directory.

STDOUT

Non-zero exit code if there is an error.

getpe

This script provides a report for all PE device names and associated IP addresses contained in the ISC database. The display is sent to the computer screen by default, or you can specify an output file, using the **-f filename** script option flag.

Command Syntax

getpe <-f filename> | getpe -help

STDOUT

Creating getpe.txt in current directory

File Name

Default is getpe.txt (found in the directory where the script was executed). An alternate file name can be specified in the input parameters.

File Output Example

```
atlnga95r11-0038|null
stlsmo95r10-0063|null
stlsmo95r11-0064|null
washdc95r10-0068|null
washdc95r11-0069|null
atlnga95r12-0051|null
nycmny95r12-0057|null
okldca95r10-0059|null
okldca95r11-0060|null
cmbrma95r11-0084|null
dllstx95r13-0062|null
lsanca95r12-0054|null
```

```
okldca95r12-0061|null
clmboh95r10-0078|135.184.109.52
clmboh95r11-0079|null
atlnga95r10-0037|null
lsanca95r10-0035|10.20.21.136
lsanca95r11-0036|null
dllstx95r10-0033|null
dllstx95r11-0034|null
chcgil95r10-0039|135.184.14.155
```

Log Name

None

modifyce

This script modifies the CE device names in the ISC database. The **inputfilename** parameter is used to specify the CE device names to be changed.

For example, the following input file:

1234 5678

4321 8765

makes these modifications:

- The site named C1234 is changed to C5678
- The device named c1234 is changed to c5678
- The site named C4321 is changed to C8765
- The device named c4321 is changed to c8765

Command Syntax

modifyce <-input filename> [-log <logFileName>]

To send the output to a log file, use the **-log** script option and specify a **logFileName**.

STDOUT

0 for success, 1 for failure.

File Name

None

Log Name

Default log name is $ISC_HOME/tmp/modifyce.log.$$

Where $$ is the UNIX process id assigned to this script when it is run. An alternate log file name can be specified in the input parameters.

Log Output Example

```
**********************************************************************
*
*
```

```
Tue Aug  3 09:19:19 PDT 2004
********Detailed log messages for each of CE and it's Site name modification****
***
Success: Site with the name C1234 changed to C4321 and it's CE name changed from
 c1234 to c4321
******All the given CE names and it's Site name changed successfully!*******

**********************************************************************
```

purgesrs

This script performs these actions:

- Finds all service requests in the ISC database that are in the *Closed* state.
- Purges or removes each of these service requests from the ISC database.

If you specify a file and filename that contains a list of service request job IDs (**SRJobId**), only the service requests listed in the file are purged, and only if they are in the *Closed* state.

To purge service requests regardless of the state use the **-force** script option flag.

Command Syntax

purgesrs [-file <filename>] [-log <logFileName>] [-force]

If no arguments are given, all service requests in the *Closed* state are purged.

Table C-3 purgesrs Command Options

Option	Description
-file <filename>	The file containing the list of service requests to be purged.
-log <logFileName>	The log output file name.
-force	All service requests in <filename> are force purged

STDOUT

None

File Name

None

Log Name

Specified on the command line.

Log Output Example

```
SR with Id 140403 was purged
```

removesr

Use this script to change a specified service request to the *Decommissioned* state. The service request remains in the ISC database but is not deployed. Use the job ID (**SRJobId**) to specify the service request to decommission.

Command Syntax

removesr SRJobId

STDOUT

New SR created 140403

File Name

None

Log name

None

showsr

This script performs these actions:

- Find all the MPLS service requests in the ISC database which are not in the *Deployed, Functional*, or *Closed* state.
- Find the VPNs associated with each MPLS service request.
- Find the PE and CE devices associated with each MPLS service request
- Display this information in a table format.

When no arguments are specified, the output lists all service requests that are not in the *Deployed, Functional,* or *Closed* state.

Command Syntax

showsr [-a] [<last_N_sr>] [sr_state]

showsr [-p pvc_id]

showsr [-v vpn_name]

Table C-4 showsr Command Options

Option	Description
-a	Prints all service requests regardless of the state.
last_N_sr	Truncates the number of service requests reported, regardless of the state.

Table C-4 showsr Command Options

Option	Description
sr_state	Reports only service requests in a specified state. Valid [sr_state] values are: REQUESTED, PENDING, FAILED_DEPLOY, INVALID, DEPLOYED, BROKEN, FUNCTIONAL, LOST, CLOSED, FAILED_AUDIT and WAIT_DEPLOY.
<last_N_sr> [sr_state]	Prints the last (N) of service requests in a specified state. If last_N_sr = 0, all service requests in state [sr_state] are printed.
-p pvc_id	Reports only service requests with a specific device ID.
-v vpn_name	Reports only service requests with a specific VPN name.

STDOUT

Job_ID SR_STATE PE_ROUTER CE_ROUTER VPN_ID CREATION_DATE_TIME

149 DEPLOYED dllstx95r10-0033 c1333698 V34 2004-1-27 15:56:18

File Name

None

Log Name

None

srdump

This script performs one of these actions:

- Returns information about all service requests in the ISC database which contain the network device specified by the **pvc_id** parameter.
- Returns information about the service request designated by the **sr_id**. The **-sr** script option is required when requesting **sr_id**.

Command Syntax

srdump <pvc_id> [-disable] [-configlet]

srdump -sr <sr_id> [-configlet]

Table C-5 srdump Command Options

Option	Description
-sr	Indicates that the required argument refers to a service request ID. If -sr is not specified, a PVC device name must be defined.
<sr_id> | <pvc_id>	The required identification number of the service request for this report. • service request ID <sr_id> • PVC device name <pvc_id>
-disable	Disables full reports. Only a brief report is displayed for each service request. Use this option to reduce the amount of data reported. This option is only available with the <pvc_id> argument.
-configlet	Prints the configlet for each service request

STDOUT

```
CREATION_TIME             2004-1-27 16:12:30
MODIFICATION_TIME         2004-1-27 16:12:41
SR_ID                     147
OpType                    ADD
SR_STATE                  DEPLOYED
CE_NAME                   c1331520.customer
PE_NAME                   nycmny95r11-0044.noc.att.com
BGP_AS                    65000
CE_ADDR                   128.222.253.118/30
CE_ENCAP                  FRAME_RELAY
CE_INTERFACE              Serial0.100
CE_DLCI/CE_VCD            777
CE_VCI                    -1
CE_VPI                    -1
NEIGHBOR_AS_OVERRIDE      false
PE_ADDR                   128.222.253.117/30
PE_ENCAP                  ATM
PE_INTERFACE              Switch1.216
PE_IF_SHUTDOWN            false
PE_VCD                    1
PE_VCI                    216
PE_VPI                    0
Vrf_Rd_Overwrite_Enabled  false
CERC                      any_to_any
IsHub                     true
HUB_RT                    13979:34
RD                        13979:34
VRF_NAME                  34
PE_CE_PROTOCOL            BGP

 Last State Change Comment: -1


------------------------------
 no  rate-limit input access-group rate-limit 6 56000 16000 32000 conform-action transmit
exceed-action set-prec-transmit 1
exit
int Switch1.216
 no  rate-limit input access-group rate-limit 7 208000 40000 80000 conform-action transmit
exceed-action set-prec-transmit 4
exit
```

```
int Switch1.216
 no   service-policy output COS_POLICY4:1
 pvc 0/216
 no   service-policy output COS_POLICY4:1
exit
int Switch1.216 point-to-point
 ip accounting precedence input
 rate-limit input access-group rate-limit 8 8000 8000 8000 conform-action
set-prec-continue 0 exceed-action set-prec-continue 0
 rate-limit input access-group rate-limit 7 8000 8000 8000 conform-action transmit
exceed-action set-prec-transmit 4
 rate-limit input access-group rate-limit 6 8000 8000 8000 conform-action transmit
exceed-action set-prec-transmit 1
 pvc 0/216
 service-policy output COS_POLICY3:1
 tx-ring-limit 3
exit
ip vrf 34
 maximum routes 4500 75
router bgp 13979
address-family ipv4 vrf 34
 default-information originate
 maximum-paths eibgp 6
 neighbor 128.222.253.118 route-map set-CE-local-pref in
exit
-------------------------------
```

taskdump

This script provides information about service request tasks. Indicate the detail of the report by specifying either a:

- service request ID (**sr_id**)
- task name (**task_name**)

Command Syntax

taskdump -h | <sr_id> | <task_name> [-verbose]

Table C-6 taskdump Command Options

Option	Description
-h	Prints the help message
sr_id	Obtain information about tasks associated with service request.
task_name	Obtain information about a specified task.
-verbose	Obtain detailed task information.

STDOUT

```
Date: 2004-08-03T09:10:41 Level: INFO Message: Open repository succeeded
 ======== Creating ProvDrvSR succeeded for Job#140418SR#140423
     Date: 2004-08-03T09:11:07 Level: INFO Message: MPLS_VPN_Link[ 140413 ] Status [[
c2571924 ] Successful Deployment<br>[ dllstx95r10-0033 ] Successful Deployment<br>]
     Date: 2004-08-03T09:11:08 Level: INFO Message: Open repository succeeded
 ======== Creating ProvDrvSR succeeded for Job#140418SR#140423
bash-2.05b$ taskdump 140418
```

```
     Date: 2004-08-03T09:10:41 Level: INFO Message: Open repository succeeded
 ======== Creating ProvDrvSR succeeded for Job#140418SR#140423
     Date: 2004-08-03T09:11:07 Level: INFO Message: MPLS_VPN_Link[ 140413 ] Status [[
c2571924 ] Successful Deployment<br>[ dllstx95r10-0033 ] Successful Deployment<br>]
     Date: 2004-08-03T09:11:08 Level: INFO Message: Open repository succeeded
 ======== Creating ProvDrvSR succeeded for Job#140418SR#140423
```

File Name

None

Log Name

None

upinterface

Use this script to turn on (or turn up) given network interface (**interfaceName**) on a given device (**rpmName**). This script logs into the specified RPM device and inserts the **no shutdown** IOS command on the specified interface.

Command Syntax

upinterface -rpm rpmName [-user userName] -pw userPassword -enableuser enableUserName -enablepw enablePassword -interface interfaceName [-log Log file name]

Table C-7 upinterface Command Options

Command Option	Description
-rpm	Host name (or IP address) of the RPM (PE device). Required parameter.
-user	Login username. This parameter is only required if both the username and password are required for login.
-pw	Login password. Required parameter.
-enableuser	Enable username. This parameter is only required if both the username and password are required to enter enable mode.
-enablepw	Enable password. Required parameter.
-interface	The complete interface name (for example, Switch1.1). Required parameter.
-log	Log filename. Optional parameter. If not specified, the file upinterface.log is created in the $ECSP_HOME/tmp directory.

STDOUT

Non-zero exit code if there is an error.

VrfPing

VrfPing checks the connectivity between the PE and CE by executing **traceroute vrf** and **ping atm** commands. If the traceroute vrf command succeeds, VrfPing returns with an exit status of 0. The ping atm command is executed only if the VCI value is specified with the **-vci** option and the traceroute command fails.

The exit states of VrfPing are:

- 0 - traceroute command successful.
- 1 - traceroute command failed. ping atm command successful (if vci was specified).
- 2 - traceroute command failed. ping atm command failed.

Command Syntax

VrfPing -pe pe_name -ce ce_name -vrf vrf_name [-vci vci_value] [-user user_name] -pw user_passwd [-enuser enable_username] -enpw enable_passwd [-log log_file_name]

Table C-8 VrfPing Options

Option	Description
-pe	Host name (or IP address) of the PE device (RPM). Required parameter.
-ce	VPN interface address of the CE device. Required parameter.
-vrf	VRF name. Required parameter.
-vci	VCI value of the ATM subinterface.
-user	Login username. Required only if both username and password are required for login.
-pw	Login password. Required parameter.
-enuser	Enable username for PE. Required only if both username and password are required for login.
-enpw	Enable password for the PE device.
-log	Log file name. This parameter is optional. If not specified, the file vrfping.log is created in the $ECSP_HOME/tmp directory.

STDOUT

Non-zero exit code if there is an error.

Script Subdirectories

These subdirectories are located in the scripts main directory.

util

This directory contains UNIX shell scripts that are used by the UNIX shell scripts in the main scripts directory. They perform utility functions which might be used by any of the UNIX shell scripts in the main directory. Users that create or modify scripts in the main directory have reference to these utility script, but they cannot be used directly or modified.

xml

This directory contains input request XML template files. The main directory UNIX shell scripts read, copy, and modify the copied XML template file to generate inputs for the ISC NBI. The files in this directory are not modified throughout the process.

filters

This directory contains variables, used by the UNIX shell scripts in the main directory, to filter the responses generated by the ISC NBI before the response data is formatted for output to the user. As you create or modify UNIX shell scripts in the main directory, you might need to modify or add new filter files to this directory.

queries

This directory contains input request XML template files, similar to those in the xml subdirectory, but these files are in a different and more detailed format. The main directory UNIX shell scripts use the files in this directory in much the same way those in the *xml* directory are used. The resulting output from the NBI API are more detailed, and the scripts using the files of this directory can generate more detailed and formatted output to present to the user.

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z