Introduction to the Prime Provisioning API

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

The Prime Provisioning 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 some of the operations that are available in the Prime Provisioning 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 Prime Provisioning services and provides examples for provisioning MPLS, L2VPN, VPLS, EVC, and TEM services in your network.

This chapter contains the following sections:

API Components

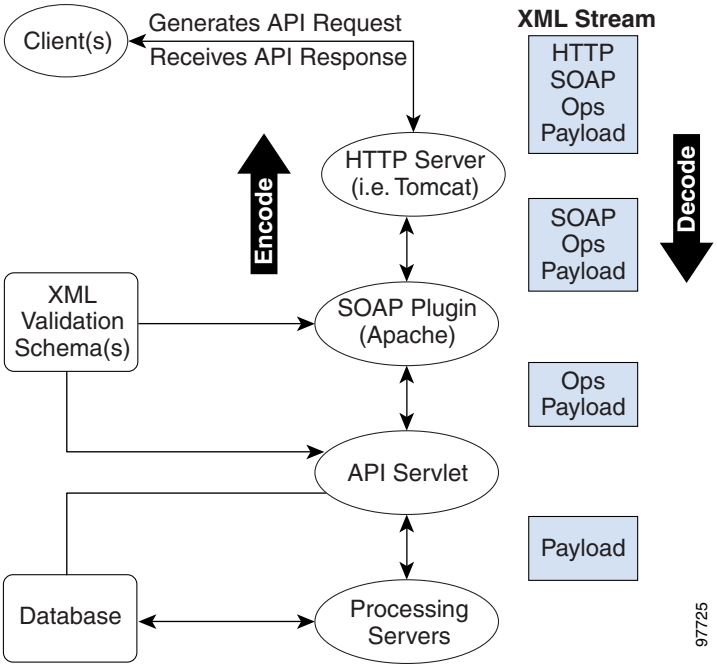
The main components of the Prime Provisioning API are:

- 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. Prime Provisioning learns of events that occur using the Tibco Event Bus.
- Processing Servers—The servers that perform the specific processing activities.
- Database—The Prime Provisioning repository.
- XML validation schema(s) and metadata—Validation files for the XML encoded data.

Figure 1-1 shows the main components of the Prime Provisioning 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 in to the Prime Provisioning 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

Prime Provisioning 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 Prime Provisioning installation. See the *Cisco Prime Provisioning 6.8 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. See the client software included with the Prime Provisioning 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 Prime Provisioning installation process. When the SSL certificate is installed on the server, you can send requests using the HTTPS protocol instead of HTTP.

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

SOAP Plug-in

Prime Provisioning 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 Prime Provisioning 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, Prime Provisioning 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 Prime Provisioning 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 indicate 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. Prime Provisioning ignores this ID.
Timestamp	Time when message was sent (in Zulu time). For more information on the date/time format, see Date/Time Format in API Requests, page 3-14.
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 Prime Provisioning 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 Prime Provisioning cannot access it for provisioning, see the "Device Locking" section on page 5-22.

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 Prime Provisioning error reporting, see the "API Error Messages" section on page 1-14.

Message Security

The Prime Provisioning 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 3-9 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 Prime Provisioning properties file and can be specified during installation.

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

API Server/Servlet

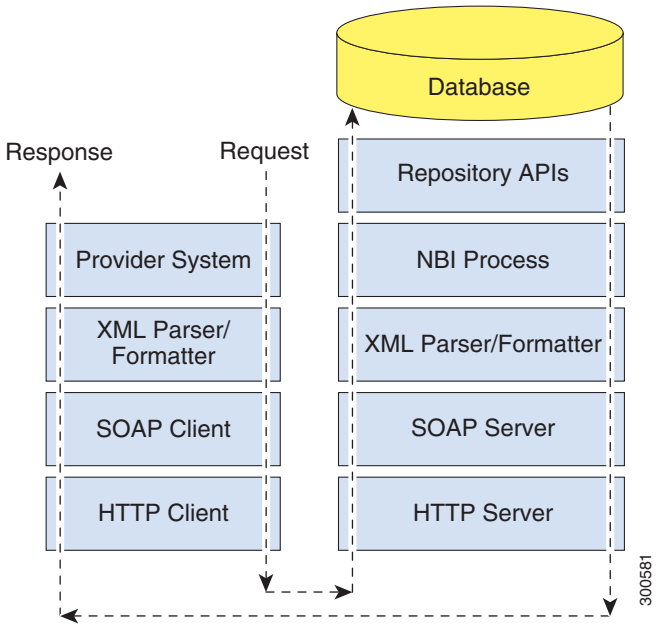
The API server (running as a servlet) receives the SOAP messages and removes the SOAP encoding. The API server also validates that the message is formatted correctly before initiating processing.

- 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 Prime Provisioning 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 Prime Provisioning inventory and service objects. The API repository object model contains all object relationships, attributes, and operations.

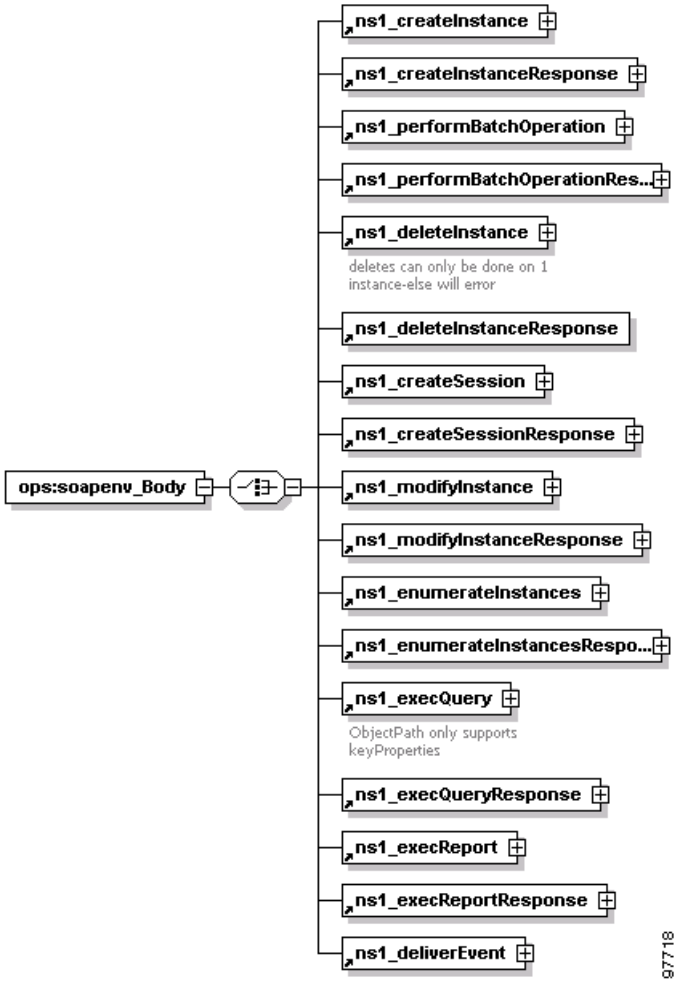
API operations are divided into three categories: general, specialized, and response.

- General operations are executed on Prime Provisioning 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 Prime Provisioning 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

Prime Provisioning 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.

Note The inventory of XML examples for the Prime Provisioning API is available at: *Cisco Prime Provisioning API 6.8 Programmer Reference*.

XML Examples

Prime Provisioning 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 Prime Provisioning API can be downloaded from here:

Cisco Prime Provisioning API 6.8 Programmer Reference

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 Prime Provisioning

Example XML Category	Described in
Evc	Chapter 9, "EVC Provisioning."
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"
Reports	Chapter 5, "Monitoring APIs"
Session	Chapter 3, "Common APIs."
SLA	Chapter 5, "Monitoring APIs"
Task	Chapter 3, "Common APIs."
TEM	Chapter 10, "Traffic Engineering Management Provisioning."
Templates	Chapter 4, "Using Templates."
VPLS	Chapter 8, "VPLS Provisioning"
MPLS-TP	Chapter 12, "MPLS Transport Profile Provisioning"
RAN Backhaul	Chapter 11, "RAN Backhaul Provisioning"

Service Model

The Prime Provisioning 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 Prime Provisioning 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

Prime Provisioning 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 Prime Provisioning 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 Prime Provisioning.

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.
- Prime Provisioning 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 4-22 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, Prime Provisioning 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 5-24.

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.

Prime Provisioning supports service definitions for each of the service types (MPLS, L2VPN), and for templates.

See the appropriate chapter on service provisioning for more information. For more information on template service definitions, see "Template Service Definitions" section on page 4-5.

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.

An error can be caused by more than one Prime Provisioning 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.

Prime Provisioning 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 error message documentation. 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).