CHAPTER 5

Monitoring APIs

Cisco Prime Fulfillment 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, page 5-1
- Reports, page 5-4
- SLA Provisioning, page 5-12
- Device Locking, page 5-22
- Viewing Task Logs, page 5-24

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 Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment 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 $PRIMEF_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 Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment product.

Note Prime Fulfillment's reporting feature supports Sybase and Oracle databases.

The Prime Fulfillment 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 5-19.

SQL-Based Reports

The **execQuery** operation allows you to execute a direct search of the Prime Fulfillment 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 Prime Fulfillment API supports a single character delimiter.

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

- The main Prime Fulfillment 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

Prime Fulfillment 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 $PRIMEF_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 $PRIMEF_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 Prime Fulfillment.

Table 5-1 Canned Reports Provided with Prime Fulfillment

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.
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 Prime Fulfillment.

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 Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment 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. Prime Fulfillment 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.

Prime Fulfillment 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. Prime Fulfillment collects the data from the appropriate SLA databases. See the "SLA Reports" section on page 5-19.
5. Prime Fulfillment 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 Prime Fulfillment database. There is one database for each collection server.

Using the Prime Fulfillment 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).

Prime Fulfillment 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 5-19 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

Table 5-2 Create SLA Probe

Operation	className	Children
	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, Prime Fulfillment 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.

Table 5-3 Common Probe Parameters (continued)

Parameter	Description
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

Prime Fulfillment 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

Table 5-4 Attributes for Probe Types

Probe Type	Attributes
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 Prime Fulfillment 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.

Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment 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>
```