CHAPTER 8

VPLS Provisioning

Cisco Prime Fulfillment 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, Prime Fulfillment 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 Prime Fulfillment 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 Prime Fulfillment API. The provisioning example includes all steps from creating the inventory to auditing the service deployment.

For more information on VPLS provisioning using Prime Fulfillment, see the *Cisco Prime Fulfillment User Guide 6.1*.

This chapter contains the following sections:

- VPLS Service Definitions, page 8-2
- VPLS Service Requests, page 8-3
- Provisioning Example, page 8-6

VPLS Service Definitions

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

Prime Fulfillment 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

A 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:

1. Create device groups (optional).
2. Create devices.
3. Collect device configurations.
4. Create provider.
5. Create regions.
6. Declare devices as PEs.
7. Create access domains and assign PEs to them.
8. Create customer.
9. Create sites.
10. Declare devices as CPEs (not required for service types with no CE present).
11. Assign CPE devices to sites.
12. Create named physical circuits (not required if **ManualConfig=true**).
13. Create VLAN ID pool.
14. Create VC ID pool.
15. Create VPN.
16. Create VPLS service definition (policy).
17. Create VPLS service request.

Prerequisites

For security reasons, Prime Fulfillment 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

Prime Fulfillment 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.

See the *Cisco Prime Fulfillment User Guide 6.1* 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 is available here:
Cisco Prime Fulfillment API Programmer Reference 6.1

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 Prime Fulfillment manages must be defined as a device in the system. An element is any device from which Prime Fulfillment 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 Prime Fulfillment database. The collection task is executed through a service request, and the service request is scheduled through a service order.

Note The service request name must be unique for each NBI API.

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.

Prime Fulfillment 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 Prime Fulfillment 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. Prime Fulfillment 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**).

Prime Fulfillment 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 4-18.

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, Prime Fulfillment 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.