Using Templates

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

Templates are defined in service definitions and can be deployed:

- Using a service order.
- Attached to a service request for another service (see the "Templates in a Service Request" section on page 4-18).

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

This chapter contains the following sections:

See the *Cisco Prime Provisioning 6.8 User Guide* for information on the GUI Template Manager.

For further information about templates from an MPLS provisioning perspective, see the template information in the MPLS part of the *Cisco Prime Provisioning 6.8 User Guide*.

Introduction to Templates in Prime Provisioning

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

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

The API supports the following types of template operations:

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

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

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

To view the interaction between the template and the device, use the task logs. See the "Viewing Task Logs" section on page 5-24 for more information.

Template Definition

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

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

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

Template Data

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

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

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

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

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

Dynamic Instantiation of Templates

A template using other templates is called a super template. The template being used by another template is called a subtemplate.

This section describes how super and subtemplate templates work and their limitations.

The super template instantiates all required subtemplates by passing values for the variables in the subtemplate. After instantiation, the super template puts the configlets generated for the subtemplate into the super template.

The subtemplate will have optional device attributes. These can be attached to a policy or a service request. During deployment, the super template will pick one of the subtemplates based on real-time link details. Prime Provisioning branches templates into subtemplates based on device type, line card type, port type, role type, and software versions. These optional attributes are set while creating the subtemplates. The subtemplates are selected based on the following matching criteria:

- Only exact matches are recognized for the card type and port type attributes. No wild card match is allowed for these attributes.
- Only an exact match is recognized for the device type attribute.
- For the software version attribute, the match is done for a software version equal to the current version, if available. If not, the previous highest version is matched.
- If none of the attributes are matched, then the default subtemplate is applied.
- If no default subtemplate exists, a subtemplate with all null attribute values is matched.
- If no match occurs, then no subtemplate is used. A warning message is displayed.

Furthermore, super templates and subtemplates are characterized by the following:

- Only one level of subtemplate is supported, but there are no checks for depth of subtemplates.
- No validations occur to check if super template and subtemplate structure is cyclic.
- When you try to delete a subtemplate that is referenced by a super-template, a warning message appears. You can modify a subtemplate.
- Subtemplates can be attached to multiple super templates.
- Only one subtemplate will be picked for a super template.
- Datafiles are not supported for subtemplates. If multiple datafiles are found, the first available datafile is chosen based on the alphabetic sorting during deployment.

An overview of available template operations is provided in Overview of Template Operations, page 4-4.

For more information on the process of selecting subtemplates in Prime Provisioning, see the *Cisco Prime Provisioning 6.8 User Guide*.

Template Operations

Prime Provisioning's template features allows you to perform a number of basic operations, policy-level operations, and service-level operations. Doing this requires a variety of subtypes, service definitions, and service orders.

This section describes the following:

Overview of Template Operations

The following template operations are available in Prime Provisioning. Examples of a selection of these operations are provided in Examples of Template Operations, page 4-6.

Basic Template Manager Functions

- Create templates and negate templates for different configurations.
- Specify device attributes for the templates.
- Associate subtemplates to templates, if applicable
- Create data files for the subtemplates.
- Create a negate template for each subtemplate.
- Create data files for the negate templates.
- Create a super template and attach subtemplates to it.

Policy-Level Template Functions

- Create a policy and enable template support for the policy.
- Associate templates to the policy, if desired.

Service-Level Template Functions

Note When a policy is only associated with a template and no data file, then during creation of a service request using that policy, the selection of a data file for that template takes place, if the template has only one data file.

- Create a service request and associate template(s) to a link.
- Deploy the service request on a device (for example, a 7600).
- The subtemplate and corresponding data file for the 7600 are autoselected for deployment.
- A configlet is generated from the subtemplate.
- Decommission the service request.
- The negate template for the subtemplate is autoselected and deployed.

Template Subtypes

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

The API supports these template subtypes:

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

The following template operations can be executed using the API:

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

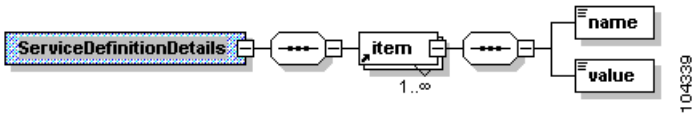
Template Service Definitions

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

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

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

Figure 4-1 Schema Diagram for Template Service Definitions



Template Service Orders

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

If the template is:

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

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

To view a template service order:

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

Examples of Template Operations

This section contains the following examples of template operations:

Adding New Templates

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

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

You are not required to create a new service order to add new templates. In one modify service request, you can turn off templates, add negate templates, and add new templates. However, you must keep the correct order of operations (turn off, add negate, then add new). This is described in more detail in Modifying a Service Request, page 4-23.

See the following example:

```
<ns1:performBatchOperation>
    <actions xsi:type="ns1:CIMActionList"
             soapenc:arrayType="ns1:CIMAction[]">
     <action>
      <actionName xsi:type="xsd:string">createInstance</actionName>
      <objectPath xsi:type="ns1:CIMObjectPath">
       <className xsi:type="xsd:string">ServiceOrder</className>
       <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">ServiceName</name>
         <value xsi:type="xsd:string">Acme-Template1</value>
        </item>
```

```
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">CarrierId</name>
    <value xsi:type="xsd:string">22</value>
   </item>
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">DesiredDueDate</name>
    <value xsi:type="xsd:dateTime">2002-12-13T14:55:38.885Z</value>
   </item>
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Organization</name>
    <value xsi:type="xsd:dateTime">NbiCustomer</value>
   </item>
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">NumberOfRequests</name>
    <value xsi:type="xsd:string">1</value>
   </item>
  </properties>
 </objectPath>
</action>
<action>
 <actionName xsi:type="xsd:string">modifyInstance</actionName>
 <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
 <className xsi:type="xsd:string">ServiceRequest</className>
 <properties xsi:type="ns1:CIMPropertyList"
             soapenc:arrayType="ns1:CIMProperty[]">
  <item xsi:type="ns1:CIMProperty">
   <name xsi:type="xsd:string">RequestName</name>
   <value xsi:type="xsd:string">Template1</value>
  </item>
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">Type</name>
    <value xsi:type="xsd:string">Mpls</value>
   </item>
  </properties>
  <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">ServiceRequestDetails</className>
       <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                 soapenc:arrayType="ns1:CIMKeyProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">LocatorId</name>
                <value xsi:type="xsd:string">36</value>
              </item>
         </keyProperties>
         <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">MplsVpnLink</className>
           <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                  soapenc:arrayType="ns1:CIMKeyProperty[]">
              <item xsi:type="ns1:CIMProperty">
                <name xsi:type="xsd:string">LocatorId</name>
                <value xsi:type="xsd:string">33</value>
              </item>
           </keyProperties>
           <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
           </properties>
           <objectPath subAction="modifyInstance" xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">LinkAttrs</className>
           <keyProperties xsi:type="ns1:CIMKeyPropertyList"
                  soapenc:arrayType="ns1:CIMKeyProperty[]">
          </keyProperties>
          <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
          </properties>
         </objectPath>
```

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

Setting Up Optional Template Attributes

In the following example, a template is created with optional device-specific attributes.

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

<item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">DeviceType</name>
    <value xsi:type="xsd:string">7600</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">CardType</name>
    <value xsi:type="xsd:string">7600-ES20-D3C</value>
  </item>
  <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">PortType</name>
    <value xsi:type="xsd:string">Gigabit12</value>
  </item>
   <item xsi:type="ns1:CIMProperty">
    <name xsi:type="xsd:string">SoftwareVersion</name>
    <value xsi:type="xsd:string">13.2.0</value>
  </item>
</properties>
</objectPath>
```

Creating Negate Templates

In Prime Provisioning, a template is removed by way of negate templates. These have to be created one time for each template and will then be selected automatically if present. No manual intervention is required. If there are no negate templates, the template commands are not removed.

In the following example, a negate template is created.

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

        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Description</name>
          <value xsi:type="xsd:string">A template definition for negate template.</value>
        </item>

        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">BodyText</name>
          <value xsi:type="xsd:string"> user $name </value>
        </item>
        </properties>
          <objectPath xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">TemplateString</className>
          <properties xsi:type="ns1:CIMPropertyList"
                      soapenc:arrayType="ns1:CIMProperty[]">
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">Dimension</name>
              <value xsi:type="xsd:string">0</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">VariableName</name>
              <value xsi:type="xsd:string">name</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">MinLength</name>
              <value xsi:type="xsd:string">5</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">Required</name>
              <value xsi:type="xsd:string">true</value>
            </item>
            <item xsi:type="ns1:CIMProperty">
              <name xsi:type="xsd:string">AvailableValues</name>
              <value xsi:type="xsd:string">admin, iscadmin</value>
            </item>
          </properties>
         </objectPath>
       </objectPath>
```

Creating Subtemplates

In the following example, a subtemplate with attributes is created.

```
<objectPath xsi:type="ns1:CIMObjectPath">
<className xsi:type="xsd:string">SubTemplate</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">/test/t4</value>
 </item>
 <item xsi:type="ns1:CIMProperty">
   <name xsi:type="xsd:string">DeviceDefault</name>
   <value xsi:type="xsd:string">false</value>
 </item>
 <item xsi:type="ns1:CIMProperty">
```

```
      <name xsi:type="xsd:string">VersionDefault</name>
      <value xsi:type="xsd:string">false</value>
    </item>
   </properties>
</objectPath>
```

Creating Templates in Policies (N-PE)

In this example, a N-PE-based policy template is configured based on a template definition and a template data file.

```
<objectPath xsi:type="ns1:CIMObjectPath">
  <className xsi:type="xsd:string">PolicyTemplate</className>
   <properties xsi:type="ns1:CIMPropertyList"
              soapenc:arrayType="ns1:CIMProperty[]">
    <item xsi:type="ns1:CIMProperty">
      <name xsi:type="xsd:string">RoleType</name>
      <value xsi:type="xsd:string">N-PE</value>
   </item>

     <item xsi:type="ns1:CIMProperty">
       <name xsi:type="xsd:string">DatafilePath</name>
       <value xsi:type="xsd:string">Certificate/Cert-Enrollment</value>
    </item>
    <item xsi:type="ns1:CIMProperty">
       <name xsi:type="xsd:string">DatafileName</name>
       <value xsi:type="xsd:string">SampleData0</value>
    </item>

    <item xsi:type="ns1:CIMProperty">
       <name xsi:type="xsd:string">TemplateAction</name>
       <value xsi:type="xsd:string">APPEND</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">Editable</name>
       <value xsi:type="xsd:string">true</value>
    </item>
     </properties>
</objectPath>
```

Creating Templates in Policies (U-PE)

In this example, a U-PE-based policy template is configured based on a template definition and a template data file. The U-PE template is differentiated by having two policy template nodes with UNIDatafilePath/UNIDatafileName for UNI and DatafilePath/DatafileName for all others.

When you include templates in a MPLS, L2VPN, VPLS or EVC service policy, the template information is defined using the PolicyTemplate object. This policy template contains the path to the template definition and the location of the template data. For each policy type, the policy template is defined in service definition details.

This is also valid for PE-AGG alone after the U-PE.

```
<objectPath xsi:type="ns1:CIMObjectPath">
    <className xsi:type="xsd:string">PolicyTemplate</className>
    <properties xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty"><name xsi:type="xsd:string">RoleType</name>
            <value xsi:type="xsd:string">U-PE</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">UNIDatafilePath</name>
            <value xsi:type="xsd:string">/nbi/AccessList2</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">UNIDatafileName</name>
            <value xsi:type="xsd:string">MyTemplate2</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>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">Editable</name>
            <value xsi:type="xsd:string">true</value>
        </item>
    </properties>
</objectPath>
<objectPath xsi:type="ns1:CIMObjectPath">
    <className xsi:type="xsd:string">PolicyTemplate</className>
    <properties xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty"><name xsi:type="xsd:string">RoleType</name>
            <value xsi:type="xsd:string">U-PE</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">DatafilePath</name>
            <value xsi:type="xsd:string">/nbi/AccessList1</value>
        </item>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">DatafileName</name>
            <value xsi:type="xsd:string">MyTemplate2</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>
        <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">Editable</name>
            <value xsi:type="xsd:string">true</value>
        </item>
    </properties>
</objectPath>
```

Creating Datafile from Service Request

In the following example, a data file is created from a service request.

```
<objectPath xsi:type="ns1:CIMObjectPath">
   <className xsi:type="xsd:string">DataFile</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">/Examples/AccessList1/Data0</value>
             </item>
             <item xsi:type="ns1:CIMProperty">
               <name xsi:type="xsd:string">Description</name>
                <value xsi:type="xsd:string">Description for datafile</value>
             </item>
             <item xsi:type="ns1:CIMProperty">
                 <name xsi:type="xsd:string">Application</name>
                 <value xsi:type="xsd:string">eq smtp</value>
             </item>
             <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">protocol</name>
                  <value xsi:type="xsd:string">tcp</value>
             </item>
        </properties>
    </objectPath>
```

Provisioning Example

This section describes the required steps for using templates independent of service requests. See the "Templates in a Service Request" section on page 4-18 for information on deploying templates with service requests.

This section describes the following:

Prerequisites

Prime Provisioning provides prepopulated examples to help you create a template.

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

Process Summary

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

1. Create a template definition file.

2. Create a template data file.
3. View the template data file.
4. View the template configuration.
5. Download the template configuration to a device.
6. Delete the template data file and template definition.

Detailed Provisioning Process

This section provides an example provisioning process using XML examples. The inventory of XML examples for the Prime Provisioning API is available here:

Cisco Prime Provisioning API 7.0 Programmer Reference

To execute the Process Summary mentioned above, use the following steps:

Step 1 Create a template definition.

Table 4-1 Create Template Definition

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

XML Example:

- CreateTemplateDefnSimple.xml

Step 2 Create a template data file.

Table 4-2 Create Template Data

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

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

XML Examples:

- CreateTemplateData1Dim.xml
- CreateTemplateData2Dim.xml

Step 3 View the template data file.

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

Table 4-3 View Template Data

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

XML Example:

ViewTemplateData.xml

Step 4 View the configlet generated for the template.

Table 4-4 View Configlet

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

XML Example:

ViewTemplateConfig.xml

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

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://www.cisco.com/cim-cx/2.0" xmlns:ns1="urn:CIM">
  <soapenv:Header>
    <ns0:message id="87855" sessiontoken="F1856684E9A183F5E542890772B3D040"
timestamp="2003-09-25T21:38:07.645Z" />
  </soapenv:Header>
  <soapenv:Body>
```

```
    <ns1:enumerateInstancesResponse>
      <returns xsi:type="ns1:CIMPropertyList" soapenc:arrayType="ns1:CIMProperty[]">
        <item xsi:type="ns1:CIMProperty">
          <name xsi:type="xsd:string">Configlet</name>
          <value xsi:type="xsd:string">
ip vrf $vrfName
 maximum routes 2 75
  export map To_NM_VPN
 route-target import 2:103000
 route-target import 2:103500
 route-target import 2:103020
 route-target import 2:103520
 route-target import 0
 route-target import 1
 exit
router bgp 13979
 address-family ipv4 vrf vrf1
 default-information originate
 maximum-paths eibgp 6
 bgp suppress-inactive
   neighbor 10.10.10.1 route-map set_ce_local_pref in
 neighbor 10.10.10.1 maximum-prefix 21 50
 neighbor 10.10.10.1 capability orf prefix-list receive
 neighbor 10.10.10.1 advertisement-interval 60
    exit
</value>
        </item>
      </returns>
    </ns1:enumerateInstancesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Step 5 Download the template configuration to a device.

Table 4-5 Download Template Configuration

Operation	className	Required Parameters
performBatchOperation		
createInstance	ServiceOrder	• ServiceName • NumberOfRequests This number indicates the number of devices to which the templates need to be downloaded. • ServiceRequest

Table 4-5 Download Template Configuration

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

XML Example:

CreateTemplateServiceOrderDownload.xml

Note For this XML, the tag <action> <actionName xsi:type="xsd:string">createInstance</actionName> </action> should be repeated for every request . For example, if the number of requests is 2, then the tag should be repeated two times in the XML with the proper inputs.

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

Table 4-6 Delete Template Files

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

XML Examples:

- DeleteTemplateData.xml

- DeleteTemplateDefn.xml

Transient Templates

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

- To view the generated configlet, refer to "View the configlet generated for the template." on page 14.
- To download transient template data to a device, refer to "Download the template configuration to a device." on page 15.

For transient templates, leave out the following two properties:

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

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

```
<objectPath xsi:type="ns1:CIMObjectPath">
      <className xsi:type="xsd:string">ServiceRequest</className>
      <properties xsi:type="ns1:CIMPropertyList"
                  soapenc:arrayType="ns1:CIMProperty[]">
       <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">RequestName</name>
        <value xsi:type="xsd:string">MYSR-1</value>
       </item>
       <item xsi:type="ns1:CIMProperty">
        <name xsi:type="xsd:string">Type</name>
        <value xsi:type="xsd:string">TemplateDownload</value>
       </item>
        <item xsi:type="ns1:CIMProperty">
         <name xsi:type="xsd:string">ServiceDefinition</name>
          <value xsi:type="xsd:string">/User/UsernameTemplate</value>
          <qualifier xsi:type="ns1:CIMQualifier">
            <name xsi:type="xsd:string">ServiceDefintionType</name>
            <value xsi:type="xsd:string">TemplateDefn</value>
          </qualifier>
        </item>
        <objectPath xsi:type="ns1:CIMObjectPath">
         <className xsi:type="xsd:string">ServiceDefinition</className>
         <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
         <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">SubType</name>
            <value xsi:type="xsd:string">TemplateDataTransient</value>
         </item>
         </properties>
         <objectPath xsi:type="ns1:CIMObjectPath">
          <className xsi:type="xsd:string">ServiceDefinitionDetails</className>
          <properties xsi:type="ns1:CIMPropertyList"
                   soapenc:arrayType="ns1:CIMProperty[]">
           <item xsi:type="ns1:CIMProperty">
            <name xsi:type="xsd:string">username</name>
            <value xsi:type="xsd:string">user1</value>
           </item>
```

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

XML Example:

CreateTemplateServiceOrderDownloadTransient.xml

Templates in a Service Request

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

To add templates to a service request, see Adding New Templates, page 4-6.

To remove templates from a service request, see "Removing Template Configurations" section on page 4-22.

Using Template Node Objects

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

This node object is SRAssociatedTemplate for EVC and LinkTemplate for other provisioning services as described in the following sections:

- Link Template
- SRAssociatedTemplate.

Link Template

When you include templates in a MPLS, L2VPN, or VPLS service request, the template information is defined using the **LinkTemplate** object. The **LinkTemplate** contains the path to the template definition and the location of the template data.

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

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

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

The following is an example of using the LinkTemplate node:

```
<objectPath subAction="createInstance" 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">8</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">DatafileName</name>
                <value xsi:type="xsd:string">MyTemplate1</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>
    </objectPath>
```

SRAssociatedTemplate

In the case of EVC, when you include templates in an EVC service request, the template information is defined using the SRAssociatedTemplate object. As with LinkTemplate, the SRAssociatedTemplate contains the path to the template definition and the location of the template data.

For the EVC service type, the SRAssociatedTemplate is defined in the **EvcLink** link object in the service request.

See Chapter 9, "EVC Provisioning" for more information on using SRAssociatedTemplate in service requests.

The following is an example of using the SRAssociatedTemplate node:

```
<objectPath subAction="createInstance" xsi:type="ns1:CIMObjectPath">
                <className xsi:type="xsd:string">SRAssociatedTemplate</className>
                <properties xsi:type="ns1:CIMPropertyList"
                                    soapenc:arrayType="ns1:CIMProperty[]">
                <item xsi:type="ns1:CIMProperty">
                     <name xsi:type="xsd:string">Device</name>
                     <value xsi:type="xsd:string">iscind-3750-6</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">TemplateDefn</name>
                      <value xsi:type="xsd:string">/Examples/AccessList1</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">TemplateData</name>
                  <value xsi:type="xsd:string">Protocol-IP</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">TemplateActive</name>
                  <value xsi:type="xsd:string">false</value>
                </item>
                <item xsi:type="ns1:CIMProperty">
                  <name xsi:type="xsd:string">TemplateActionType</name>
                  <value xsi:type="xsd:string">APPEND</value>
                </item>
               </properties>
              </objectPath>
```

Data Buffer Object

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

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

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

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

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

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

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

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

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

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

Table 4-7 Templates in a Service Request

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

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

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

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

XML Examples:

- CreateMPLSTemplateServiceOrder.xml
- CreateL2VPNTemplateServiceOrder.xml

Removing Template Configurations

To modify a service request that has templates and before decommissioning a service request that has templates, first remove the template information from the service request. This is accomplished by applying a negate template to the existing service, which is done automatically in Prime Provisioning.

Using Negate Templates

You can assign both a positive and negative template/data file to a policy. Prime Provisioning can determine which one it should use based on the requested service request action (for example, deploying or decommissioning a service).

The negate template has the same name as the template, with the addition of the suffix .Negate. The negate template uses the same data file as the positive template on which it is based.

Modifying a Service Request

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

1. Turn off templates.

   This action changes the **TemplateActive** attribute for the template from **true** to **false**.

   - For MPLS service requests, the **modifyInstance** subaction automatically toggles the **TemplateActive** attribute to **false**.

     See Turning off Templates (for MPLS Service Requests), page 4-23.

   - For all other service requests, the **TemplateActive** attribute must be specifically set to **false** in the **modifyInstance** subaction.

     See Turning off Templates (for All Other Service Types), page 4-24.

2. Add new templates.

   This action adds a new template to the service request. Use the **createInstance** action to add templates.

   See Adding New Templates, page 4-6.

Note Negate templates are added automatically at the time when the template is selected. As a result, you do not need to attach a negate template to a policy or a service request.

Note Wait until the task has completed (you will receive a task completed message) before running the next task.

Turning off Templates (for MPLS Service Requests)

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

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

Note This automatic change in the template attribute only occurs when you use a **modifyInstance** on a template in an MPLS service request. For other service type, see the "Turning off Templates (for All Other Service Types)" section on page 4-24.

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

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

```
                    <item xsi:type="ns1:CIMProperty">
                      <name xsi:type="xsd:string">DatafilePath</name>
                      <value xsi:type="xsd:string">/Examples/templ1-enable</value>
                    </item>
```

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

Turning off Templates (for All Other Service Types)

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

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