Managing Dynamic File Configuration

This chapter describes the following features that Prime Cable Provisioning supports for device configuration and device management:

Groovy Scripting

This section explains the Groovy scripting support that Prime Cable Provisioning provides for device configuration and device management. This section features:

Overview

Prime Cable Provisioning uses Groovy scripting, apart from templates, for generating the configuration file, which helps you to deploy dynamic files for any CableLabs standard supported by Prime Cable Provisioning including DOCSIS, PacketCable, CableHome, and OpenCable STB. This scripting interface allows you to access the discovered DHCP data and device properties, which will help in deciding the TFTP file that has to be generated. The Prime Cable Provisioning RDU generates the configuration file using either the template or Groovy scripting. The RDU identifies the Groovy file by the extension, .groovy. Groovy sample script files are available in the BPR_HOME/rdu/samples/groovy directory, which can be used for testing.

To create your Groovy script file, you should be familiar with the Groovy scripting language, in addition to the requirements for templates creation.

Groovy Script Language

A Groovy script can include the following options:

Groovy Script

Option	Description	Example
<comment>	// [ascii-string] /* * Multi-line comments */	// Config File Start/End / * configFile—of type DOCSISTFTPFile * services—of type ExtensionServices * discoveredData—of type DHCPDataAccess * deviceProperties—of type CSRCProperties * option—of type DOCSISoptionfactory * device—of type IPDevice * context—of type ConfigContext */
<option-description>	<option-with-no-suboptions> | <compound-option>	
<option-with-no- sub options>	configFile.add(option.createOptionValue (<custom-value>, "<option-num>","<option-value>"));	configFile.add(option.createOptionValue ("3", "1")); For custom values: configFile.add(option.createOptionValue (OptionSyntaxHEX., "217.53","010868446146484A4737"));

<compound-option>	def <variable-name> = option
	.createOptionValue("<option-num>");
	<variable-name>.add(option
	.createOptionValue(<custom-value>,
	"<optionnum>","<option-value>"));
	configFile.add(<variable-name>);
		def option24 = option.createOptionValue("24");
		option24.add(option.createOptionValue("24.8", "4194304"));
		configFile.add(option24);

<custom-value> optional	OptionSyntax.HEX | OptionSyntax.ASCII | OptionSyntax.SNMP	

Option	Description	Example
<option-num>	<unsigned-byte>[.<unsigned-byte>]*	"24"
<option-value>	<option-value-string> [,<option-value-string>] *	"1" or [".docsDevNmAccessCommunity.1", "Octet String", "private"] as String[]

Bindings that are visible to the Groovy environment are:

- configFile—of type DOCSISTFTPFile
- services—of type ExtensionServices
- discoveredData—of type DHCPDataAccess
- deviceProperties—of type CSRCProperties
- option—of type DOCSISoptionfactory
- device—of type IPDevice
- context—of type ConfigContext

Note Device object binding is not available while executing the Groovy script from CLI File Utility.

Note To avoid any compilation error while creating groovy scripts, use:

- single quotes (' ') for the string.

  For example: `configFile.add(option.createOptionValue(OptionSyntax.SNMP,"11", ['.iso.org.dod.internet.private.enterprises.8595.2.1.2.10.1.2.2','STRING','msopassword']));`

- double quotes (" ") for the string, with (\) before any special character.

  For example: `configFile.add(option.createOptionValue(OptionSyntax.SNMP,"11", ['.iso.org.dod.internet.private.enterprises.8595.2.1.2.10.1.2.2','STRING',"ms\$opassword"]));`

Adding a Groovy Script to Prime Cable Provisioning RDU

To add a Groovy script file to a Prime Cable Provisioning RDU:

Step 1 Choose **Configuration > Files**. The View Files page appears.

Step 2 Click **Add**. The Add Files page appears.

Step 3 Choose the CableLabs Configuration Script option from the **File Type** drop-down list.

Step 4 Browse for the Source File Name

Step 5 Add the <*filename*>*.groovy* file in the File Name field.

Step 6 Click **Submit**.

Using Configuration File Utility for Groovy

Configuration file utility is used to convert groovy file to a binary configuration file and vice versa. It can also be used to view and validate the configuration and groovy files. The configuration file utility is installed in the BPR_HOME/rdu/bin directory. The groovy file and the binary file must be available in the directory from where the configuration file utility is invoked.

Note Since Prime Cable Provisioning uses the configuration utility only to generate binary to groovy file and vice versa, it will not support other scripting languages.

This section discusses the following topics:

Running the Configuration File Utility

To run the configuration file utility, run the command from the *BPR_HOME/rdu/bin* directory:

runCfgUtil.sh options

The available options include:

- **-?**—Prints this usage message.
- **-e**—Performs encoding of a BACC groovy/template file (default).
- **-d**—Performs decoding of a binary file.
- **-g**—Performs generation of a groovy/template file from a binary file.
- **-snmp**—Performs generation of a template or dynamic script file from a binary file with OptionSyntax.SNMP enabled.
- **-c** *shared*—The CMTS shared secret to use when parsing a BACC groovy/template file (the default is cisco).
- **-h** *host:port*—Specifies where the RDU is located (the default is localhost:49187).
- **-i** *device ID*—Specifies the device to use for macro variables substitutions when parsing a groovy/template.
- **-m** *macros*—Specifies the macro variables to be substituted when parsing a groovy/template.
- **-s**—Displays the parsed groovy/template or the contents of the binary file in a human readable format.
- **-o** *filename*—Saves the parsed groovy/template or the human readable output in the specified filename.
- **-l** *filename*—Specifies the input file to be on the local file system.
- **-r** *filename*—Specifies the input file to be remote on the RDU.
- -**pkt**—Specifies the file to be processed as a PacketCable MTA configuration file.
- **-t** *type*—Specifies the PacketCable encoding type (default is secure).
- **-loc** *locale*—Specifies the PacketCable locale such as na, euro, and, ietf (default is na). The default is na. If the MTA is euro-MTA, then the locale should be set to euro.
- **-cablehome**—Specifies the file to be processed as a CableHome configuration file.
- **-docsis**—Specifies the file to be processed as a DOCSIS configuration file (default).
- **-E**—Enable Extended CMTS MIC (EMIC) calculation and identifies the default options for EMIC calculation. The default options are:
    - HMAC type—MMH16
    - EMIC Digest type—Explicit
    - EMIC shared secret as cisco.
- **-Ei**—Specifies [implicit] presentation that will be used for Extended CMTS MIC Digest Subtype.
- **-Eh** *HMACType*—Specifies the hashing algorithm used to compute Extended CMTS MIC. The supported algorithms are MD5 and MMH16 (default is MMH16).
- **-Es** *secret*—The CMTS shared secret to use for Extended CMTS MIC calculation (the default is cisco).
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*—Specifies the password to use when connecting to the RDU.

- **-v** *version*—Specifies the version of the technology to process the input file.
- **-prop** *filename*—Specifies the property file that has the key and value for the variables used in dynamic script.

Note You should always specify either **-DDV4** or **-DDV6** filename along with **-prop** filename to pass Discovered data.

- **-dis** *filename*—Specifies the discovered data to be used in the dynamic script in the form key and value pair.
- **-DDv4** *filename*—Specifies the discovered DHCPv4 data to be used in dynamic script in the form key and value pair.
- **-DDv6** *filename*—Specifies the discovered DHCPv6 data to be used in dynamic script in the form key and value pair.
- **-cp** *classpath*—Specifies the path of the extension jars and script files referred in dynamic script.
- **-b**—Specifies bulk processing option for generating multiple output files. All the binary files in the given directory (using -l option) will be processed and the generated files will be available in the output directory indicated in -o option.
- **-ft**—The file type (groovy or tmpl)to be generated. This option will be used when bulk processing is enabled (using -b option) for generation(-g option) operation. (The default file type is tmpl.)

Validating a Groovy Script Using runCfgUtil

To use the configuration file utility to test Prime Cable Provisioning Groovy script:

Step 1 Develop the Groovy script. If the Groovy script extends to other Groovy scripts, make sure all the referenced Groovy scripts are in the same directory.

Step 2 Run the configuration file utility on the local file system. You can check the syntax for the Groovy script, or have the configuration file utility process the Groovy script as CRS would, and return output.

If the Groovy script contains dynamic variables or the discovered data, perform these operations in the order specified:

a) Test with command line substitution with property file.
b) Test with a device that has been added to your RDU.

Step 3 Add the Groovy script (and any extended Groovy scripts that are used) to the RDU.

Step 4 Run the configuration file utility to parse a file. See Testing Groovy Script Processing for an External Groovy Script File.

If the Groovy script contains dynamic variables or the discovered data, perform these operations in the order specified:

a) Test with command line substitution with property file.
b) Test with a device that has been added to your RDU.

Step 5 After all tests succeed, configure a Class of Service to use the Groovy script.

Converting a Binary File to a Groovy Script File

Use the runCfgUtil.sh command to convert binary configuration memory files into Groovy script files. Prime Cable Provisioning dynamic configuration generation is based on Groovy scripts that are created. Automatically converting existing, tested, binary files to Groovy script files speeds the process and reduces the possibility of introducing errors.

Note Using the **runCfgUtil.sh** tool, you cannot convert a template directly into Groovy scripts and vice versa. You must first convert the template into a binary file, and then convert the binary file into a Groovy script. When you convert a Groovy script to template, you must first convert the Groovy script into a binary file, and then convert the binary file into a template.

Syntax Description

runCfgUtil.sh -g -l *binary_file* **-o** *groovy_file*

- **-g**—Specifies that a Groovy script file needs to be generated from an input binary file
- **-l** *binary_file*—Specifies the local input file, including the pathname; bronze.cm for example.
- **-o** *groovy_file*—Specifies the output Groovy script file, including the pathname. In all cases, the output Groovy script file will have a .groovy file extension; for example, test.groovy.

To convert a binary file into a Groovy script file:

Step 1 Change directory to */opt/CSCObac/rdu/samples/docsis*.

Step 2 Select a Groovy script file to use. This example uses an existing binary file called *unprov.cm*.

Step 3 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -g -l unprov.cm -o test.groovy -docsis
```

-docsis—Specifies that the input file is a DOCSIS configuration file.

Converting a Binary File to a Groovy Script File Without Dependency on MIBs

Use the **runCfgUtil.sh** command with the option -snmp to convert binary configuration memory files into Groovy script files without dependency on the MIBs. When this command is executed with the -snmp option, it adds **OptionSyntax.SNMP** for all TLVs that contain SnmpVarBind. Ensure that you specify the correct value based on the type for TLV, for example, integer numbers (1).

Note The OID must be in numeric format to completely remove the MIB dependency, for example, .1.3.44491.1.2.3.

Syntax Description

runCfgUtil.sh -docsis -g -snmp -l *binary_file* **-o** *groovy_file*

- **-g**—Identifies the input file as a PacketCable MTA file.

- **-snmp** —Specifies that the generated dynamic script file from the binary file has OptionSyntax.SNMP enabled.
- **-l** *binary_file*—Specifies the local input file, including the pathname. In all cases, the input binary filename will have a .cm file extension; bronze.cm for example.
- **-o** *groovy_file*—Specifies the output Groovy script file, including the pathname. In all cases, the output Groovy script file will have a .groovy file extension; for example, test.groovy

To convert a binary file into a Groovy script file without dependency on MIBs:

Step 1 Back up the MIBs.

Step 2 Add the property `/snmp/mibs/mibList=` to api.properties located in the directory `BPR_Home/api/conf/`.

Note While generating device configuration from RDU, add the property `/snmp/mibs/mibList=` to rdu.properties located at `BPR_HOME/rdu/conf` file to avoid MIB dependency.

Step 3 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -g -snmp -l example.cm -o example.cm.groovy
```

- **example.cm**—Identifies the input binary file.
- **example.cm.groovy**—Identifies converted groovy file.

Testing Groovy Script Processing for a Local Groovy Script File

Use the **runCfgUtil.sh** command to test the processing for Groovy script files stored on the local file system.

Syntax Description

runCfgUtil.sh -pkt -l *file*

- **-pkt**—Identifies the input file as a PacketCable MTA file.
- **-l**—Specifies that the input file is on the local file system.
- *file*—Identifies the input Groovy script file being parsed.

To parse a Groovy script file that is on the local file system:

Step 1 Change directory to */opt/CSCObac/rdu/samples/packet_cable*.

Step 2 Select a Groovy script file to use. This example uses an existing Groovy script file called *unprov_packet_cable.groovy*. The **-pkt** option is used because this is a PacketCable MTA Groovy script.

Step 3 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -pkt -l unprov_packet_cable.groovy
```

unprov_packet_cable.groovy—Identifies the input Groovy script file being parsed.

Testing Groovy Script Processing for an External Groovy Script File

Use the **runCfgUtil.sh** command to test processing of external Groovy script files.

Syntax Description

runCfgUtil.sh -docsis -r *file* **-u** *username* **-p** *password*

- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the input Groovy script file being parsed.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.
- **-docsis**—Identifies the file as a DOCSIS Groovy script.

To parse a Groovy script file that has been added to the RDU:

Step 1 Select a Groovy script file to use. This example uses an existing Groovy script file called *unprov.groovy*. The **-docsis** option is used because a DOCSIS Groovy script is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -docsis -r unprov.groovy -u admin -p changeme
```

- **unprov.groovy**—Identifies the input file.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

Testing Groovy Script Processing for a Local Groovy Script File and Adding Shared Secret

Use the **runCfgUtil.sh** command to test the processing for a Groovy script file and add a shared secret that you specify.

Syntax Description

runCfgUtil.sh -e -docsis -l *file* **-c** *shared*

- **-e**—Identifies the encode option.
- **-docsis**—Identifies the input file as a DOCSIS Groovy script file.
- **-l**—Specifies that the input file is on the local file system.
- *file*—Identifies the input Groovy script file being parsed.
- **-c**—Specifies the CMTS shared secret when parsing a DOCSIS Groovy script file.
- *shared*—Identifies the shared secret. The default shared secret is **cisco**.

To parse a locally saved Groovy script file, and set a user-specified shared secret:

Step 1 Change directory to */opt/CSCObac/rdu/groovy*.

Step 2 Select a Groovy script file to parse. This example uses an existing Groovy script file called *unprov.groovy*. The **-docsis** option is used because this is a DOCSIS Groovy script.

Step 3 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -docsis -l unprov.groovy -c shared
```

- **unprov.groovy**—Identifies the input file on the local file system.
- **shared**—Identifies that shared secret.

Testing Groovy Script Processing for a Local Groovy Script File and Adding EMIC Shared Secret

Use the **runCfgUtil.sh** command to test the processing for a Groovy script file and add a Extended CMTS MIC (EMIC) shared secret that you specify.

Syntax Description

runCfgUtil.sh -E -docsis -l *filename*

- **-E**—Enables EMIC calculation.
- **-docsis**—Identifies the input file as a DOCSIS Groovy script file.
- **-l** *filename*—Specifies the input Groovy script file, including the pathname. In all cases, the input Groovy script file will have a *.groovy* file extension; for example, test.groovy.

To calculate the EMIC with default settings:

Step 1 Select a Groovy script file to use. This example uses an existing Groovy script file called *unprov.groovy*. The **-docsis** option is used because a DOCSIS Groovy script is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -E -l test.groovy
```

Specifying Dynamic Variables at the Command Line

Use the **runCfgUtil.sh** command to specify dynamic variables.

Syntax Description

runCfgUtil.sh -e -l *file* **-prop** *“file”*

- **-e**—Identifies the encode option.
- **-l**—Specifies the input file is on the local file system.
- *file*—Identifies the input Groovy script file being parsed.
- **-prop**—Specifies the property file that has key and value for variables used in dynamic script.

- *"file"*—Identifies the desired dynamic variable. If multiple dynamic variables are required, then each key value pair should be given one after the other.

To specify values for dynamic variables at the command line:

Step 1 Change directory to */opt/CSCObac/rdu/groovy*.

Step 2 Select a Groovy script file to use.

Step 3 Identify the dynamic variables in the Groovy script.

Step 4 Identify the values for the variables.

Step 5 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -l macro.groovy -prop prop.properties
```

- **macro.groovy**—Identifies the input file.
- **prop.properties**—Contains key value and pair (eg: MTA_PROP=3)

Specifying a Device for Dynamic Variables

Use the **runCfgUtil.sh** command to specify a device for dynamic variables.

Syntax Description

runCfgUtil.sh -e -r *file* **-i** *MAC* **-u** *username* **-p** *password*

- **-e**—Identifies the encode option. Accepts key and if not mentioned, it takes the default key.
- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the input Groovy script file being parsed.
- **-i**—Specifies the device to use when parsing dynamic variables.
- *MAC*—Identifies the MAC address of the device.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.

To specify a device to be used for dynamic variable substitution:

Step 1 Select a Groovy script file to use. This example uses the existing Groovy script file, *macro.groovy*.

Step 2 Identify the dynamic variables in the Groovy script.

Step 3 Identify the device to use. This example assumes that the device exists in the RDU and has the dynamic variables set as properties.

Step 4 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -r macro.groovy -i "1,6,00:01:02:03:04:05" -u
admin -p changeme
```

Note When you use -i option to specify the device MAC, you must also mention the encode option (-e) and input file option (-r) with it.

- **macro.groovy**—Identifies the input file.
- **1,6,00:01:02:03:04:05**—Identifies the MAC address of the device. The MAC address used here is an example only.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

Specifying Discovered Data at the Command Line

Use the **runCfgUtil.sh** command to specify Discovered Data.

Syntax Description

runCfgUtil.sh -e -l *file* **-dis** *"file"*

- **-e**—Identifies the encode option. Accepts key and if not mentioned, it takes the default key.
- **-l**—Specifies the input file is on the local file system.
- *file*—Identifies the input Groovy script file being parsed.
- **-dis**—Specifies the discovered data to be used in dynamic script in the form key and value pair.
- *"file"*—Identifies the desired discovered data.

To specify values for discovered data at the command line:

Step 1 Select a Groovy script file to use.
Step 2 Identify the discovered data in the Groovy script.
Step 3 Identify the values for the discovered data.
Step 4 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -l macro.groovy -dis dis.properties
```

- **macro.groovy**—Identifies the input file.
- **dis.properties**—contains key value and pair (eg: giaddr=10.1.1.9).

Specifying a Device for Discovered Data

Use the **runCfgUtil.sh** command to specify a device and use its discovered data for configuration file generation.

Syntax Description

runCfgUtil.sh -e -r *file* **-i** *MAC* **-u** *username* **-p** *password*

- **-e**—Identifies the encode option. Accepts key and if not mentioned, it takes the default key.

- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the input Groovy script file being parsed.
- **-i**—Specifies the device to use when parsing discovered data.
- *MAC*—Identifies the MAC address of the device.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.

To specify a device to be used for discovered data substitution:

Step 1 Select a Groovy script file to use. This example uses the existing Groovy script file, *macro.groovy*.

Step 2 Identify the discovered data in the Groovy script.

Step 3 Identify the device to use. This example assumes that the device exists in the RDU and has the discovered data set as properties.

Step 4 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -r macro.groovy -i "1,6,00:01:02:03:04:05" -u
admin -p changeme
```

- **macro.groovy**—Identifies the input file.
- **1,6,00:01:02:03:04:05**—Identifies the MAC address of the device. The MAC address used here is an example only.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

Generate Binary File from Groovy

Use the **runCfgUtil.sh** command to specify the output of a parsed Groovy script as a binary file.

Syntax Description

runCfgUtil.sh -l *input_file* **-o** *output_file*

- **-l**—Specifies that the input file is on the local file system.
- *input_file*—Identifies the input Groovy script file being parsed.
- **-o**—Specifies that the parsed Groovy script file is to be saved as a binary file.
- *output_file*—Identifies the name of the file in which the binary contents of the parsed Groovy script file are stored.

To specify the output from parsing a Groovy script to a binary file:

Step 1 Select a Groovy script file to use.

Step 2 Identify the name of the output file. This example uses *unprov.cm*.

Step 3 Run the configuration file utility using this command:

/opt/CSCObac/rdu/bin# **`runCfgUtil.sh -l unprov.groovy -o unprov.cm`**

- **`unprov.groovy`**—Identifies the existing Groovy script file being parsed into a binary file.
- **unprov.cm**—Identifies the output filename to be used.

Viewing a Local Binary File

See Viewing a Local Binary File for details.

Viewing an External Binary File

See Viewing an External Binary File for details.

Activating PacketCable Basic Flow

See Activating PacketCable Basic Flow, on page 61 for details.

Generating TLV 43s for Multivendor Support

See Generating TLV 43s for Multivendor Support for details.

Dynamic TFTP File-Naming Convention

The TFTP File-Naming Convention helps you customize the variable components of the dynamic TFTP filenames, and their order. The Groovy script generates the TFTP filename by using components like the DHCP discovered data as well as the other interfaces that are being exposed to it. The script can include any important information such as, the class of service name, discovered vendor name, downstream speed and so on. You can configure the script either in technology defaults or in system defaults. The default maximum filename length is 127 characters.

If a CableLabs configuration filename script is modified, configuration regeneration is triggered for the list of affected devices to reflect the changes.

You can find Groovy script samples in *BAC_HOME/rdu/samples/groovy*.

Example 19-1 Sample TFTP Filename Groovy Script

```
/**
* example_extended_filename.groovy
*
* A sample CableLabs Configuration Filename Script that demonstrates how
* to create an extended filename. This example includes the following
* strings in the extended filename: DeviceType, Selected ClassOfService,
* and provisioning group. For DOCSIS device types, the default DOCSIS
* version is included after device type. The resulting extended filename
* string is:
*
* "<device-type>_<default-docsis-version>_<selected-cos>_<pg>"
* (e.g., "cm_11_goldcos_westpg", "pc_silvercos_eastpg").
*
* A CableLabs Configuration Filename Script specifies an extended filename
* label that is appended to the standard BAC dynamic configuration filename.
* In BAC 4.2 and later releases, the dynamic configurations have a filename
* consisting of the fixed/standard prefix. The script can be configured at
* System Defaults (preferred) and/or Technology Defaults.
*
* BAC properties:
* DocsisDefaultKeys.DOCSIS_DEFAULT_VERSION
*
* Variable bindings:
* configFileName - Extended Filename of type StringBuilder
* services - of type ExtensionServices
* discoveredData - of type DHCPDataAccess
* deviceProperties - of type CSRCProperties
* device - of type IPDevice
* context - of type ConfigContext
*/
import com.cisco.provisioning.cpe.constants.DocsisDefaultKeys
import com.cisco.provisioning.cpe.extensions.constants.CNRNames
import com.cisco.provisioning.cpe.extensions.services.DeviceType
/*
* A Groovy list is used to collect the ordered list of string fields
* that will comprise the extended filename. Once all the fields have been
* added to the list, the "join" method is used to concatenate the
* fields with a underscore ('_') separator character.
*/
def label = []
/*
* Add Device Type (abbreviated).
*
* The device type string is too verbose for a filename component, so an
* abbreviation is used instead. For example, the DOCSIS device type value
* "DOCSISModem" is abbreviated as "cm". If no abbreviation is defined,
* a default abbreviation of "xx" is used.
*/
def deviceType = device.getDeviceType().getName()
def deviceTypeMap = [
(DeviceType.DOCSIS_MODEM) : "cm",
(DeviceType.PACKET_CABLE_MTA) : "pc",
(DeviceType.CABLEHOME_WAN_MAN) : "ch",
(DeviceType.STB) : "st",
(DeviceType.CUSTOM_CPE): "cu"
]
label << deviceTypeMap[deviceType] ?: "xx"
/*
* Add default DOCSIS version number (exclude embedded "dot").
*
* For DOCSIS device types, the default DOCSIS version number specifies the
* maximum DOCSIS version supported by the CM and CMTS. This version number
* indicates the DOCSIS version grammar used when constructing the dynamic
* configuration file. The embedded "dot" is stripped from the version number
* (i.e., "3.0" --> "30").
*/
if (deviceType == DeviceType.DOCSIS_MODEM)
{
label << deviceProperties.getProperty(
DocsisDefaultKeys.DOCSIS_DEFAULT_VERSION, "1.0") - "."
}
/*
* Add Selected Class of Service name.
*/
label << device.getSelectedClassOfService().getClassOfServiceName()
/*
* Add Provisioning Group name.
*/
label << device.getProvGroup().getProvGroupId()
```

```
/*
* Convert the list of filename components into a string value with underscore
* ('_') characters separating the filename components. Add the resulting
* string to the configFileName StringBuilder binding.
*/
configFileName << label.join("_")
```

Dynamic TFTP File-Naming via Extensions

The TFTP File-Naming ability has been enhanced to support customization at the RDU extension level. Prior to Prime Cable Provisioning 5.2 release, the TFTP filename can be named dynamically only using the TFTP filename generation Groovy scripts. Prime Cable Provisioning 5.2 release allows user to set the configuration filename in a shared context which is available across the service-level extensions, configuration generation extensions and configuration generation scripts.

The filename set in the shared context can be configured at technology or system defaults level which will have the higher precedence over the filename set at the TFTP filename generation Groovy scripts.

Shared Context Filename

A shared context `( com.cisco.provisioning.cpe.extensions.configuration.SharedConfig )` is now included in the ConfigContext (com.cisco.provisioning.cpe.extensions.configuration.ConfigContext) which is shared across the service-level extensions, configuration generation extensions and configuration generation Groovy scripts. The new shared context object has a setter method `( setConfigFileName (String) )` which can be used to populate the filename. The sample code snippet given below explains the filename population in shared context on extension.

```
SharedConfig sharedConfig = configContext.getSharedConfig();

sharedConfig.setConfigFileName("<fillup_the_filename_here>");
```

In addition to set and get methods for the filename, the shared context object also has a map object (can be accessed using the get method <Map<String, Object> getSharedConfigMap()) which can be used as a container to save data and share between extensions .

The shared context is included in device detection context `(com.cisco.provisioning.cpe.extensions.detection.DeviceDetectionContext)` which is used by the device detector extensions. The shared context that is populated by device detection extensions will be pre-populated in the configuration context that is shared by the service level extensions and configuration generation extensions. If the shared context is populated by the device detection extension, it will be pre-populated in the configuration context which can be used by the service level selection and configuration generation extensions.

The Device disruptor context `(com.cisco.provisioning.cpe.extensions.disruption.DeviceDisruptionContext)` now encapsulates a shared context. However, this is not relevant to the configuration context or the file naming. The shared context (com.cisco.provisioning.cpe.extensions.configuration.SharedContext) included in the device disruption context can be used by multiple device disruptor extensions.

Basic flow of Dynamic TFTP filename generation

The following is the sequence of steps that explains the basic flow of filename generation:

1. RDU receives a configuration generation request.

2. The RDU runs the configuration generation for the device, during which, the generation extension determines that a dynamic TFTP file needs to be assigned to the device. The file could be a template (for example, docsis.tmpl) or a script (for example, docsis.groovy), (for example, silver.cm)
3. After running all the custom extensions, the configuration engine looks for the availability of the filename in the shared context. If the filename is populated in the shared context, and the Provisioning Group capability "*/provgroup/capability/dpe/tftp/filename/extensions" (ProvGroupCapabilitiesKeys.TFTP_DYNAMIC_FILENAME_USING_EXTENSIONS)* is enabled, then it will follow the step 5. If the filename is not populated in the shared context, then it will follow the filename generation as per the Extended TFTP filename Groovy script workflow, as mentioned in step 4.
4. The generation extension looks for the CableLabs Configuration Filename Script property in the technology defaults. If not found, it looks in the system defaults. The value of this property is the name of the script to be executed. The script is executed and it returns the additional strings to be included in the dynamic TFTP filename.
5. The generation extension uses this value and creates a dynamic TFTP filename for the device. After the filename generation is complete, the configuration is sent to the DPEs, which is then cached.

Basic pointers about filename generation via shared context

- Like the TFTP filename Groovy script functionality, the name populated in the shared context will have the dynamic portion of the actual TFTP configuration filename
- The default maximum filename length is 127 characters and the filename will be truncated if it exceeds the limit. The space and special character removal are as same as the existing TFTP dynamic filename generation feature.
- If the filename was not populated in the shared context by any of the custom extensions or the configuration generation scripts, then the filename generation will follow the existing TFTP filename generation via configuration filename generation Groovy scripts workflow.

Sample Service-Level extension to populate the filename in Shared Context

```
package com.cisco.provisioning.cpe.extensions.samples;

import java.util.Map;

import com.cisco.provisioning.cpe.extensions.ExtensionException;
import com.cisco.provisioning.cpe.extensions.configuration.ConfigContext;
import com.cisco.provisioning.cpe.extensions.configuration.ServiceLevel;
import com.cisco.provisioning.cpe.extensions.configuration.SharedConfig;
import com.cisco.provisioning.cpe.extensions.services.ExtensionServices;
import com.cisco.provisioning.cpe.extensions.services.IPDevice;

/**
* This class will demonstrate usage of Shared Context and the filename
* availability in the configuration context object. The PCP extensions will
* make use of the Shared Context and also have the feasibility of setting the
* filename from the extensions.
*
*/
public class SampleSharedContextServiceLevelSelectionExtension implements
com.cisco.provisioning.cpe.extensions.configuration.ServiceLevelSelector
{
@Override
public void selectServiceLevel(IPDevice device,
ConfigContext configContext, ServiceLevel serviceLevel,
ExtensionServices extensionServices) throws ExtensionException
{
final String extensionName =
SampleSharedContextServiceLevelSelectionExtension.class.getName();
com.cisco.provisioning.cpe.extensions.services.LogManager logManager =
extensionServices.getLogManager();
logManager.log(
com.cisco.provisioning.cpe.extensions.services.LogLevel.INFO,
extensionName, " Executing custom code here..");
long duration = System.currentTimeMillis();
/*
* Get the SharedConfig object from ConfigContext
*/
SharedConfig sharedConfig = configContext.getSharedConfig();
/*
* Get the sharedContext map from SharedConfig object
*/
Map<String, Object> sharedContextMap =
sharedConfig.getSharedConfigMap();
/*
* Set the generated token into the Shared Context map
*/
sharedContextMap.put("customToken_1", generateTokenXYZ());
logManager.log(
com.cisco.provisioning.cpe.extensions.services.LogLevel.INFO,
extensionName, " Shared Context Map after token generation "
+ sharedContextMap);
/*
* Setting the file name in SharedConfig object
*/
logManager.log(
com.cisco.provisioning.cpe.extensions.services.LogLevel.INFO,
extensionName, " Settinng the file name here...");
sharedConfig.setConfigFileName("SetBy" + this.getClass().getCanonicalName());
/*
*
* Set a token that is used by sample configuration script
* example_shared_context.groovy
*/
sharedContextMap.put("maxCPE","7");
logManager.log(
com.cisco.provisioning.cpe.extensions.services.LogLevel.INFO,
extensionName, " Shared Context Map after new token "
+ sharedContextMap);
logManager.log(
com.cisco.provisioning.cpe.extensions.services.LogLevel.INFO,
extensionName, " It took ["
+ (System.currentTimeMillis() - duration)
+ "] milliseconds to complete this extension ");
}
/**
* This method models a custom token generation
* @return the generated token as String
*/
private String generateTokenXYZ()
{
/*
* operation.
*/
return "dummyTokenValue";
}
}
¢
Sample Configuration Script to access Shared Context and populate filename in Shared Context
```

```
/**
* A sample CableLabs Configuration Script that demonstrates the Shared Context
availability in Groovy script. And also it demonstrate to populate the filename in Shared
Context
*
*
* PCP properties:
* DocsisDefaultKeys.DOCSIS_DEFAULT_VERSION
* SNMPPropertyKeys.READ_COMMUNITY_STRING
* SNMPPropertyKeys.WRITE_COMMUNITY_STRING
*
* Variable bindings:
* configFile - of type DOCSISTFTPFile
* option - of type DOCSISOptionFactory
* services - of type ExtensionServices
* discoveredData - of type DhcpDataAccess
* deviceProperties - of type CSRCProperties
* device - of type IPDevice
* context - of type ConfigContext
*
* @see com.cisco.provisioning.cpe.extensions.configuration.DOCSISTFTPFile
* @see com.cisco.provisioning.cpe.extensions.configuration.DOCSISOptionFactory
* @see com.cisco.provisioning.cpe.extensions.services.ExtensionServices *
* @see com.cisco.provisioning.cpe.extensions.configuration.DhcpDataAccess
* @see com.cisco.provisioning.cpe.extensions.services.CSRCProperties
* @see com.cisco.provisioning.cpe.extensions.services.IPDevice
* @see com.cisco.provisioning.cpe.extensions.configuration.ConfigContext
*/
import com.cisco.provisioning.cpe.constants.SNMPPropertyKeys
import com.cisco.provisioning.cpe.extensions.configuration.SharedConfig
def TLV = option.&createOptionValue

/* 1. Accessing Shared Context */
/*
* This script tries to set the maxCPE value based on the tokens available at the Shared
Context
* Populate the maxCPE property in Shared Context by using extensions to use this
demonstration.
/*
* Get the SharedConfig object from ConfigContext
*/
SharedConfig sharedConfigObj = context.getSharedConfig();
/*
* Get the sharedContext map from SharedConfig object
*/
Map<String, Object> sharedContextMapObj =
sharedConfigObj.getSharedConfigMap();
def maxCPE = TLV("18", "3")
if (sharedContextMapObj.containsKey("maxCPE"))
{
maxCPE = TLV("18", (String)sharedContextMapObj.get("maxCPE"))
}
configFile.add(maxCPE)

/* 2. Filename can also be set at this configuration script by using the Shared Context
*/
/** Sets the dynamic filename */
sharedConfigObj.setConfigFileName("samplenamesetByCfgScript");
```

Templates

This section details the templates that Prime Cable Provisioning supports for device configuration and device management. This section features:

- Template Files–An Overview, on page 20
- Template Grammar
  - SNMP VarBind
  - Macro Variables
  - SNMP TLVs
  - Encoding Types for Defined Options
- Using Configuration File Utility for Template

Template Files–An Overview

Prime Cable Provisioning uses templates to help you deploy dynamic PacketCable, DOCSIS, and CableHome files. Using templates, you can create a template file in an easily readable format, and edit it quickly and simply. A template is an ASCII text file that represents the PacketCable, DOCSIS, or CableHome options and values used for generating a valid PacketCable, DOCSIS, or CableHome file. Prime Cable Provisioning uses the *.tmpl* extension to identify template files. You must add template files to the RDU as a file using the administrator user interface or the application programming interface (API), before any Class of Service can reference it.

When installing the Prime Cable Provisioning RDU component, several sample template files are copied to the *BPR_HOME/rdu/templates* directory.

Although all that you need to create or edit a template is a simple text editor, before attempting to create your own template file, you should thoroughly familiarize yourself with this information:

- Flow of provisioning Cisco Prime Cable Provisioning
- DOCSIS 1.0, 1.1, 2.0, 3.0, and 3.1 RFI specifications
- DOCSIS Layer 2 Virtual Private Networks specification
- PacketCable 1.0, 1.5 and 2.0 specifications
- Multimedia Terminal Adapter (MTA) device provisioning specification
- CableHome 1.0 specification
- SNMP MIBs for cable devices (for example, DOCS-CABLE-DEVICE-MIB)

Template Grammar

A template comprises the following types of statements:

- Comments, on page 21

Comments allow you to document your templates. Includes allow you to create building block templates to be used in other templates. You use options to specify the PacketCable, DOCSIS, or CableHome type length value (TLV) in a descriptive manner. You can use instance modifiers to group compound options into specific individual TLVs. The OUI modifier allows you to include vendor-specific information. The following table describes the available template grammar options.

Table 1: Template Grammar

Option	Description
<comment>	::= #[ascii-string]
<include>	::= include “<filename.tmpl>”
<option-description>	::= option <option-num> [instance <instance-num>] [oui <oui>] <option-value>
<option-num>	::= <unsigned-byte>[.<unsigned-byte>]*
<option-value>	::= <well-defined-value> | <custom-value>
<well-defined-value>	::= <option-value-string>[,<option-value-string>]*
<custom-value>	::= <ascii-value> | <hex-value> | <ip-value> | <snmp-value>
<ascii-value>	::= ascii <ascii-string>
<hex-value>	::= hex <hex-string>
<ip-value>	::= ip <ip-string>
<instance-num>	::= <unsigned integer>
<template>	::= <template-statement>*
<template-statement>	::= <comment> | <include> | <option-description>
<snmp-value>	::=<snmpvar-oid>,<snmpvar-type>,<snmpvar-value>

Comments

Comments provide information only and are always located between the pound (#) symbol and the end of a line. The following example shows sample comment usage.

Sample Comment Usage

```
#
```

```
# Template for gold service
#

option 3 1 # enabling network access
```

Includes

Include files let you build a hierarchy of similar, but slightly different, templates. This is very useful for defining options that are common across many service classes without having to duplicate the options in several templates.

You can use multiple include statements in a single template, although the location of the include statement in the template is significant: The contents of the include file are included wherever the include statement is found in the template. The included template must be added as a file to the RDU before it can be used. The included file must not contain any location modifiers such as ../.. because the templates are stored without path information in the RDU database. Example 19-3 and Example 19-4 illustrate both correct and incorrect usage of the include option.

Correct Include Statement Usage

```
# Valid, including common options
include "common_options.tmpl"
```

Incorrect Include Statement Usage

```
# Invalid, using location modifier
include "../common_options.tmpl"

# Invalid, using incorrect file suffix
include "common_options.common"

# Invalid, not using double quotes
include common_options.tmpl
```

Options

PacketCable, DOCSIS, and CableHome configuration files consist of properly encoded option ID-value pairs. Two forms of options are supported: defined and custom.

- Well-defined options require the option number and value. The value is encoded based on the encoding type of the option number.
- Custom options require the option number, explicit value encoding type, and the value.

When using compound options, for example, Option 43, you can use the instance modifier to specify the TLV groupings. See Instance Modifier, on page 23.

When specifying one of these well-defined options in a template, it is not necessary to specify a value encoding for the value. For additional information on these defined encoding types, see Encoding Types for Defined Options, on page 32, and Technology Option Support.

When specifying custom options (for example, Option 43), you must specify the encoding type for the option. The available encoding types are:

- ASCII— ASCII type encodes any given value as an ASCII string without a NULL terminator. If the value contains spaces, they must be enclosed in double quotation marks.

- hex—The value must be valid hexadecimal and there must be exactly 2 characters for each octet. If 01 is specified as the value, then exactly one octet is used in the encoding. If 0001 is specified as the value, then exactly two octets are used in the encoding process.
- IP address—IP address type encodes any given value as 4 octets. For example, the IP address 10.10.10.1 is encoded as 0A0A0A01.
- SNMPVarBind—An SNMP OID string, type, and value. Each of these is comma separated.

Use a comma to separate multivalued options on a given line. Each value is treated separately, so you might have to enclose one of the values in double quotation marks, but not the others. A good example of a multivalued option is Option 11 (SNMP VarBind). See SNMP VarBind, on page 25, for additional information.

When specifying compound options, there is no need to specify the top-level option (for example Option 4 when specifying Option 4.1). **Correct Option Statement Usage** and **Incorrect Option Statement Usage** illustrate both correct and incorrect usage of the option statement.

Correct Option Statement Usage

```
# Valid, specifying the number for well known option 3
option 3 1

# Valid, specifying the number for option 4 sub-option 1
option 4.1 1

# Valid, specifying a vendor option as hex
option 43.200 hex 00000C

# Valid, specifying a vendor option as ascii
option 43.201 ascii "enable log"

# Valid, specifying a vendor option as IP
option 43.202 ip 10.4.2.1
```

Incorrect Option Statement Usage

```
# Invalid, using hex with incorrect hex separator
option 43.200 hex 00.00.0C

# Invalid, not using double quotes when needed
option 43.201 ascii enable log

# Invalid, not specifying IP address correctly
option 43.202 ip 10-10-10-1

# Invalid, specifying the description for option "Network Access Control"
option "Network Access Control" 1

# Invalid, specifying top level option
option 4
```

Instance Modifier

The instance modifier is used to group compound options into specific individual Type-Length-Values (TLVs). Example and Example illustrate both correct and incorrect methods of creating separate TLVs. These are required to enable the IOS DOCSIS modem to interpret the IOS commands as two separate commands.

Example: Correct IOS Command Line Entries

```
# Valid, each IOS command gets its own TLV
option 43.8 instance 1 00-00-0C
option 43.131 instance 1 ascii "login"
option 43.8 instance 2 00-00-0C
option 43.131 instance 2 ascii "password cable"
```

Example: Incorrect IOS Command Line Entries

```
# Invalid, IOS commands are grouped into one TLV
option 43.8 00-00-0C
option 43.131 ascii "login"
option 43.131 ascii "password cable"

# Invalid, using instance on non-compound options
option 3 instance 1 1
```

Note The encoding type for Option 43.8 is an organizationally unique identifier (OUI). Unlike that shown in Example, this type only accepts an 00-00-0C format.

OUI Modifier

The OUI modifier enhances multi-vendor support using Option 43 and its suboptions.

In Prime Cable Provisioning, you can use a single template to specify various TLV 43s from many vendors. The example Correct OUI Modifier Usage specifies the OUI formats as XX-XX-XX, where:

- FF-FF-FF—Identifies the vendor ID to specify encoding for the DOCSIS general extension.
- 00-00-0C—Identifies the Cisco vendor ID that specifies the Cisco-specific cable modem Option 43 and its suboptions.

The example Correct OUI Modifier Usage illustrates Prime Cable Provisioning support for L2VPN using a cable modem configuration file to classify upstream traffic for L2VPN. Using this template content, you can generate subTLVs:

- 43.5.1 and 43.5.2.2 from the DOCSIS general extension encoding, using OUI=FF-FF-FF.
- 43.1 from the Cisco-specific Option 43, using OUI=00-00-0C.

However, in order to comply with the DOCSIS specification, you must insert as the first subTLV for TLV 43 either:

- 0xFFFFFF when using the DOCSIS extension field to encode general extension information.
- 0x00000C when generating Cisco-specific subTLVs.

Correct OUI Modifier Usage

```
# Upstream L2VPN Classifier Example

# This example shows how to classify upstream traffic from a specific CPE
# onto an upstream L2VPN service flow, in which other CPE attached to
# the cable modem forward to the non-L2VPN forwarder, as depicted below.

# This example also demonstrates that when using the DOCSIS extension
# field (TLV 43) to encode general extension information (GEI), you do
```

```
    # not need to specify oui=FF-FF-FF. You only need to specify the OUI tag when
    # general extension encoding is not used and vendor-specific encoding is used.

    # Upstream L2VPN Classifier Cable Modem Config File

    # (43) Per-CM L2VPN Encoding
    # GEI (43.8) Vendor ID : 0xFFFFFF for GEI
    option 43.8 instance 1 ff-ff-ff

    # GEI (43.5) for L2VPN Encoding
    # GEI (43.5.1) VPNID Subtype
    option 43.5.1  instance 1 0234560003

    # GEI (43.5) for L2VPN Encoding
    # GEI (43.5.2) IEEE 802.1Q Format Subtype
    # VLAN ID 25
    option 43.5.2.2 instance 1 25

    # Cisco Specific Vendor Option Encodings
    # (43.8) Vendor ID : 00-00-0C (Cisco Vendor ID)
    option 43.8 instance 2 00-00-0C

    # Cisco Vendor Specific option (43.1)
    # Static Downstream Frequency
    # Frequency 402750000
    option 43.1 instance 2 oui 00-00-0C 402750000

    # Cisco Specific Vendor Option Encodings
    # (43.8) Vendor ID : 00-00-0C (Cisco Vendor ID)
    option 43.8 instance 3 00-00-0C

    # Cisco Vendor Specific option (43.3)
    # Update Boot Monitor Image
    # image name (boot_monitor_image.bin)
    option 43.3 instance 3 oui 00-00-0C boot_monitor_image.bin
```

The following examples illustrate incorrect usage of the OUI modifier:

Example 1

```
# Invalid, OUI tag needs to be present for each 43 suboption if/when general extension
# encoding is not used and vendor-specific encoding is used.

option 43.8 00-00-0C

option 43.3 boot_monitor_image.bin
```

Example 2

```
# Invalid, when both OUI and instance modifier are used in authoring a template, # "instance"
 modifier needs to occur before "oui" modifier.

option 43.8 instance 1 00-00-0C

option 43.3 oui 00-00-0C instance 1 boot_monitor_image.bin
```

SNMP VarBind

You must use an object identifier (OID) when specifying DOCSIS Option 11, PacketCable Option 64, or CableHome Option 28. The MIB that contains the OID must be in one of the following MIBs loaded by the

RDU. You must specify as much of the OID as needed to uniquely identify it. You can use the name or the number of the OID. The RDU automatically loads these MIBs:

- SNMPv2-SMI
- SNMPv2-TC
- CISCO-SMI
- CISCO-TC
- SNMPv2-MIB
- RFC1213-MIB
- IANAifType-MIB
- IF-MIB

eRouter MIBs

ipNetToPhysicalTable [RFC 4293]	**IP-MIB**
vacmAccessTable [RFC 3415]	SNMP-VIEW-BASED-ACM-MIB
vacmSecurityToGroupTable [RFC 3415];	
vacmViewTreeFamilyTable [RFC 3415];	
vacmAccessReadViewName [RFC 3415];	
vacmAccessWriteViewName [RFC 3415];	
snmpCommunityTable [RFC 3584];	SNMP-COMMUNITY-MIB
snmpTargetAddrTMask [RFC 3584];	
snmpTargetAddrExtTable [RFC 3584];	
snmpTargetAddrTable [RFC 3413]	SNMP-TARGET-MIB
snmpTargetAddrTAddress [RFC 3413];	
esafeErouterInitModeControl [eDOCSIS].	**ESAFE-MIB.mib(upgrade)**

DOCSIS MIBs

These DOCSIS MIBs are loaded into the RDU:

- DOCS-IF-MIB
- DOCS-BPI-MIB
- CISCO-CABLE-SPECTRUM-MIB
- CISCO-DOCS-EXT-MIB
- SNMP-FRAMEWORK-MIB
- DOCS-CABLE-DEVICE-MIB

- CISCO-CABLE-MODEM-MIB

Note In Cisco BAC 4.1, the DOCSIS MIB, DOCS-CABLE-DEVICE-MIB-OBSOLETE (experimental branch) are removed from the RDU default loaded MIBs list since it predates the DOCS-CABLE-DEVICE-MIB (mib2 branch).

In Cisco BAC 4.2, the CL-SP-MIB-CLABDEF-I02-020920 and DOCS-BPI2-MIB-ipcdn-08 MIBs are removed from the RDU default loaded MIBs list since they are the duplicates of CLAB-DEF-MIB and DOCS-BPI2-MIB, respectively.

The references to any fully qualified MIB OIDs from the above removed MIBs should be replaced with the appropriate OIDs from the new MIBs in customer templates and scripts. However, the custom MIB option can be used to include these experimental OIDs. For more details about adding custom MIBs, see Adding SNMP TLVs With Vendor-Specific MIBs.

PacketCable MIBs

These PacketCable (North American) MIBs are loaded into the RDU:

- CLAB-DEF-MIB
- PKTC-MTA-MIB
- PKTC-SIG-MIB
- PKTC-EVENT-MIB

CableHome MIBs

These CableHome MIBs are loaded into the RDU:

- CABH-CAP-MIB
- CABH-CDP-MIB
- CABH-CTP-MIB
- CABH-PS-DEV-MIB
- CABH-QOS-MIB
- CABH-SEC-MIB

These additional MIBs are needed but are not part of the Prime Cable Provisioning product:

- CABH-CTP-MIB needs RMON2-MIB, TOKEN-RING-RMON-MIB
- CABH-SEC-MIB needs DOCS-BPI2-MIB.

Macro Variables

Macro variables are specified as values in templates that let you specify device-specific option values. When a macro variable is encountered in the template, the properties hierarchy is searched for the macro variable name and the value of the variable is then substituted. The variable name is a custom property, which is predefined in the RDU. It must not contain any spaces.

After the custom property is defined, it can be used in this property hierarchy:

- Device properties
- Provisioning Group properties
- Class of Service properties
- DHCP Criteria properties
- Technology defaults, such as PacketCable, DOCSIS, or CableHome
- System defaults

The template parser works bottom up when locating properties in the hierarchy (device first, then the Class of Service, and so on) and converts the template option syntax. The following syntax is supported for macro variables:

- ${var-name}—This syntax is a straight substitution. If the variable is not found, the parser will generate an error.
- ${var-name, ignore}—This syntax lets the template parser ignore this option if the variable value is not found in the properties hierarchy.
- ${var-name, default-value}—This syntax provides a default value if the variable is not found in the properties hierarchy.

The examples Correct Macro Variables Usage and Incorrect Macro Variables Usage illustrate correct and incorrect usage of Option 11.

Correct Macro Variables Usage

```
# Valid, using macro variable for max CPE's, straight substitution
option 18 ${MAX_CPES}

# Valid, using macro variable for max CPE's, ignore option if variable not found
# option 18 will not be defined in the DOCSIS configuration file if MAX_CPES
# is not found in the properties hierarchy
option 18 ${MAX_CPES, ignore}

# Valid, using macro variable for max CPE's with a default value
option 18 ${MAX_CPES, 1}


# Valid, using macro variable for vendor option
option 43.200 hex ${MACRO_VAR_HEX}

# Valid, using macro variable for vendor option
option 43.201 ascii ${MACRO_VAR_ASCII}

# Valid, using macro variable for vendor option
option 43.202 ip ${MACRO_VAR_IP}
```

```
# Valid, using macro variable in double quotes
option 18 "${MAX_CPES}"

# Valid, using macro variable within a value
option 43.131 ascii "hostname ${HOSTNAME}"

# Valid, using macro variables in multi-valued options
option 11 ${ACCESS_CONTROL_MIB,
.mib-2.docsDev.docsDevMIBObjects.docsDevNmAccessTable.docsDevNmAccessEntry.docsDevNmAccessControl.1},
 Integer, ${ACCESS_CONTROL_VAL, 3}

# Valid, using macro variable in an include statement
include "${EXTRA_TEMPLATE}"
# Valid, using macro variable in an include statement with a default value
include "${EXTRA_TEMPLATE, modem_reset.tmpl}"

# Valid, using macro variable in an include statement with a default value
include "${EXTRA_TEMPLATE, modem_reset}.tmpl"

# Valid, using macro variable in an include statement with an ignore clause
include "${MY_TEMPLATE, ignore}"
```

Incorrect Macro Variables Usage

```
# Invalid, using macro variable as the option number
option ${MAX_CPES} 1

# Invalid, using macro variable with space in name
option 18 ${MAX CPES}
```

SNMP TLVs

Prime Cable Provisioning supports SNMP TLVs in dynamic template files, using Option 11 and 64, for:

- DOCSIS—From Prime Cable Provisioning for Cable version 2.0 onwards.
- PacketCable—From Prime Cable Provisioning version 2.5 onwards.
- CableHome—From Prime Cable Provisioning version 2.6 onwards.

To validate the syntax of the SNMP TLVs in these template files, Prime Cable Provisioning requires a MIB file containing the corresponding SNMP OID that is referenced in the SNMP TLV. If a template contains an SNMP TLV with an SNMP OID that cannot be found in a MIB, the SNMP TLV generates a syntax error.

The following sections describe how you can add SNMP TLVs without a MIB or with a vendor-specific MIB.

Adding SNMP TLVs Without a MIB

You can add SNMP TLVs in dynamic configuration files (DOCSIS, PacketCable, CableHome) without requiring the MIB be loaded by the RDU. From within RDU configuration extensions, the functionality can be accessed with the DOCSISOptionFactory interface, using the following method:

```
public OptionValue createOptionValue(OptionSyntax syntax, String optionNumStr,
String[] optionValueList)
```

The public OptionSyntax.SNMP enumerated value can be used in the above method, in conjunction with the optionValueList containing the tuple: OID, Type, Value.

From RDU dynamic configuration templates, the following syntax is used to specify SNMP TLVs that are not validated against the RDU MIBs:

```
option option-number snmp OID, Type, Value
```

Examples:

```
# DOCS-CABLE-DEVICE-MIB:
option 11 snmp .docsDevNmAccessIp.1,IPADDRESS,192.168.1.1

# Arris vendor specific SNMP TLV (OID numbers only, mix names/numbers)
option 11 snmp .1.3.6.1.4.1.4115.1.3.1.1.2.3.2.0, INTEGER, 6
option 11 snmp .enterprises.4115.1.3.1.1.2.3.2.0, INTEGER, 6

# NOTE: trailing colon required for single octet
option 11 snmp .1.3.6.1.2.1.69.1.2.1.6.3, STRING, 'c0:'
```

The following table describes the allowed SNMP variable type names.

Table 2: SNMP Variable Types

IETF standard SMI Data Type	SNMP API name
Integer32	INTEGER
Integer (Enumerated)	INTEGER
Unsigned32	UNSIGNED32
Gauge32	GAUGE
Counter32	COUNTER
Counter64	COUNTER64
Timeticks	TIMETICKS
OCTET STRING	STRING
OBJECT IDENTIFIER	OBJID
IpAddress	IPADDRESS
BITS	STRING

For example, to specify an SMI Integer32 type, the following types are accepted (regardless of case sensitivity): Integer32, INTEGER.

For OCTET STRING type, all of the following types are accepted: OCTET STRING, OCTETSTRING, or STRING.

The custom SNMP TLV template option can be used to specify any SNMP TLV, including those that are present in the RDU MIBs. The custom SNMP TLV error checking is less stringent, and does not detect incorrect scalar/columnar references (for example, .0 versus .n in OID names).

Adding SNMP TLVs With Vendor-Specific MIBs

Adding a MIB to the RDU enables templates to use the human-readable SNMP OID while also permitting macro variables to be used with the SNMP TLV value.

If you have the MIB corresponding to the SNMP OID that you want to use, you can add the MIB file to the Prime Cable Provisioning RDU. After you add the MIB, any SNMP TLV using an SNMP OID referenced in the new MIB is recognized.

To add a new MIB to the Prime Cable Provisioning RDU:

Step 1 Launch the Prime Cable Provisioning administrator user interface.

Step 2 On the navigation bar, click **Configuration > Defaults**.

Step 3 On the Configure Defaults page that appears, click the System Defaults link on the left pane.

Step 4 In the MIB List field, paste the content of the new MIB at the end.

Step 5 Click **Submit**.

Debugging the MIB Load Order

Typically, vendors provide several MIBs requiring a specific load order to satisfy inter-MIB dependencies. But because the vendor frequently does not provide the correct load order, you must determine the correct load order yourself. This section describes how you can use Prime Cable Provisioning debugging information to resolve MIB load-order issues.

Note The MIB load order in Prime Cable Provisioning is set by the order in which the MIBs are listed in the */snmp/mibs/MibList* property

You can use the runCfgUtil.sh tool to determine the correct load order for the property specified in the *api.properties* file. The runCfgUtil.sh tool resides in the *BPR_HOME/rdu/bin* directory.

Step 1 Configure runCfgUtil.sh via the *api.properties* file using configuration content similar to that described in this step. The *api.properties* file enables Prime Cable Provisioning tracing to direct MIB debugging information to the user console.

```
#
# Enable logging to the console
#
/server/log/1/level=Info
/server/log/1/properties=level
/server/log/1/service=com.cisco.csrc.logging.SystemLogService
/server/log/1/name=Console
#
# Enable trace categories
#
/server/log/trace/rduserver/enable=enabled
#
# The list of MIBs to be added.
#
/snmp/mibs/MibList=arrishdr.mib,arris_cm_capability.mib,arris_mta_device.mib,arris_sip.mib,arris_cm.mib,
pp.mib,blp2.mib,dev0.mib,docs_evnt.mib,qos.mib,test.mib,usb.mib,snmpv2_conf.mib,rfc1493.mib,rfc1907.mib,
rfc2011.mib,rfc2013.mib,rfc2233.mib,rfc2571.mib,rfc2572.mib,rfc2573.mib,rfc2574.mib,rfc2575.mib,rfc2576
.mib,rfc2665.mib,rfc2669.mib,rfc2670.mib,rfc2786.mib,rfc2851.mib,rfc2933.mib,rfc 3083.mib
```

Step 2 With runCfgUtil.sh so configured, run the tool to encode any template containing an Option 11 or Option 64 (SNMP encoding). The tool attempts to load the MIBs specified within /snmp/mibs/MibList, and directs the complete debugging information, along with any MIB load errors, to the user console.

Step 3 Use the error information to massage the MIB order specified within /snmp/mibs/MibList until the complete set of MIBs loads without error and the file encode succeeds.

Step 4 Once you determine a successful load order, complete the procedure described in this step based on the Prime Cable Provisioning version you are using:

1. From the administrator user interface, click **Configuration > Defaults**, then the System Defaults link.
2. In the MIB List field, copy the load order information.

   The RDU is now configured to encode templates using the vendor-supplied MIBs.

Note You do not need to restart the RDU.

Ensure that you use the */snmp/mibs/mibList* string in the *api.properties* file and the MIB List field.

Encoding Types for Defined Options

The following table identifies the options with defined encoding types.

Table 3: Defined Option Encoding Types

Encoding	Input	Examples

Authorization Action	Unsigned 8-bit integer or description string.
	To allow authorization, the values are:
	• 0
	• permit
	To deny authorization, the values are:
	• 1
	• deny
		`0`
		`1`
		`permit`
		`deny`

Encoding	Input	Examples

Access View Control	Unsigned 8-bit integer or description string.
	To include the SNMPv3 Access View subtree from access view, the values are:
	• 1
	• included
	To exclude the SNMPv3 Access View subtree from access view, the values are:
	• 2
	• excluded
		`1`
		`2`
		`included`
		`excluded`

Access View Type	Unsigned 8-bit integer or description string.
	To enable read-only access, the values are:
	• 1
	• Read-only
	To enable read-write access, the values are:
	• 2
	• Read-write
		`1`
		`2`
		`Read-only`
		`Read-write`

ActInact	Unsigned 8-bit integer or description string.
	To disable the TLV, the values are:
	• 0
	• Inactive
	To enable the TLV, the values are:
	• 0
	• Active
		`0`
		`1`
		`Inactive`
		`Active`

BitFlag8	Unsigned 8-bit integer. The output is a hexadecimal string representation of the value.	`0xFE`

Encoding	Input	Examples
BitFlag32	Unsigned 32-bit integer. The output is a hexadecimal string representation of the value.	`0xFFFF0000`

Boolean	0 for false and 1 for true.	`0`
		`1`

Byte16	16 bytes specified as a hexadecimal string of 32 characters. Is typically used to represent the MIC option of the cable modem and the CMTS. No 0x prefix is allowed.	None. Prime Cable Provisioning automatically calculates the hash for the cable modem and CMTS MIC option.
Bytes	A series of hexadecimal octets. Each octet must be 2 characters.	`000102030405060708`

CPE Access Control	Unsigned 8-bit integer or description string.
	To disable device access control, the values are:
	• 0
	• Disabled
	To enable device access control, the values are:
	• 1
	• Enabled
		`0`
		`1`
		`Disabled`
		`Enabled`

DSCClassifier	Unsigned 8-bit integer or DSCClassifier string name. The unsigned integers include: • 0—DSC Add Classifier • 1—DSC Replace Classifier • 2—DSC Delete Classifier	`0`

Encoding	Input	Examples

EnableDisable	Unsigned 8-bit integer or description string.
	To disable, the values are:
	• 0
	• Disabled
	To enable, the values are:
	• 1
	• Enabled
		`0`
		`1`
		`Disabled`
		`Enabled`

Erouter Init Mode	Unsigned 8 bit integer describing the eRouter initialization mode 0: Disabled 1: IPv4 Protocol Enabled 2: IPv6 Protocol Enabled 3: Dual IP Protocol Enabled	0 1 2 3 Disabled IPv4 IPv6 Dual
eRouter Initialization Mode Override	Unsigned 8 bit integer to override the eRouter Initialization mode 1 = Ignore eRouter Initialization Mode TLV and keep the eRouter Disabled 0 = Follow eRouter Initialization	0 1 Follow eRouter Initialization - Mode eRouter - Disabled

Inet Address Peer	A one-byte InetAddressTypeCode:
	• 1 for IPv4
	• 2 for IPv6
	This value is followed by an IPv4 or an IPv6 internet address.
	As a result, this length is 5 bytes (1+4) for IPv4 and 17 bytes (1+16) for IPv6.
		`1,10.112.125.111`
		`2,0:0:0:0:0:ffff:8190:3426`

IP address	Four unsigned integer 8, dot (.) separated.	`10.10.10.1`

Encoding	Input	Examples
IPv6 address	A string representation of an IPv6 address *x:x:x:x:x:x:x:x,* where the *x*s are one to four hexadecimal digits of the eight 16-bit pieces of the address.	`2001:db8:0:0:8:800:200c:417a`

IPv4 or IPv6 address	A string representation of an IPv4 or IPv6 address.	`10.112.125.111`
		`0:0:0:0:0:ffff:8190:3426`

IP Mode	Unsigned 8-bit integer or description string.
	For IPv4 mode, the values are:
	• 0
	• IPv4
	For IPv6 mode, the values are:
	• 1
	• IPv6
		`0`
		`1`
		`IPv4`
		`IPv6`

Multiple IP addresses	Comma-separated list of IP addresses.	`10.11.12.13,10.11.12.14`
Multiple IPv6 addresses	Comma-separated list of IPv6 addresses.	`2001:db8:0:0:8:800:200c:417a,ff01:0:0:0:0:0:0:101`

MAC address	Six hexadecimal octets, colon (:) or dash (-) separated. Each octet must be exactly 2 characters. Colons and dashes must not be mixed.	`00:01:02:03:04:05`
		`00-01-02-03-04-05`

MAC address and mask	Twelve octets, colon (:) or dash (-) separated. Each octet must be 2 characters. Colons and dashes must not be mixed. The first six octets represent the MAC address; the last six represent the mask for the MAC address.	`00:01:02:03:04:05:06:07:08:09:0A:0B`
		`00-01-02-03-04-05-06-07-08-09-0A-0B`

NoLV	Type only. Does not include value or length.	`null`
NVTASCII	An ASCII string. The encoded string will not be NULL terminated.	`This is an ASCII string`

Encoding	Input	Examples
OID	An SNMP OID string.	`sysinfo.0`
OIDCF	An SNMP OID string and an unsigned integer (0 or 1), comma separated.	`sysinfo.0,1`

OnOff	Unsigned 8-bit integer.
	To switch on the TLV, the values are:
	• 0
	• On
	To switch off the TLV, the values are:
	• 1
	• Off
		`0`
		`1`
		`On`
		`Off`

OUI	Three hexadecimal octets, colon (:) or dash (-) separated. Each octet must be 2 characters.	`00-00-0C`

RFC868Time	Unsigned 32-bit integer representing the RFC868 time. The output is a date-time string that uses this format: *MM/dd/yyyy HH:mm:ss*.	`0`
		(representing "12/31/1899 19:00:00")
		`4294967295`
		(representing "02/07/2036 01:28:15")

Encoding	Input	Examples
ServiceFlow	Unsigned 8-bit integer or a service flow description string. The output is a service flow that indicates: • 0—Reserved • 1—Undefined (Dependent on CMTS implementation) • 2—Best Effort • 3—Non-real-time polling service • 4—Real-time polling service • 5—Unsolicited grant service with activity detection • 6—Unsolicited grant service	`0`

Encoding	Input	Examples
SNMPVarBind	An SNMP OID string, type, and value. Each of these is comma separated. Valid types are: • BITS • Counter • Counter32 • Counter64 • Gauge • Gauge32 • INTEGER • Integer32 • IpAddress • OCTETSTRING • OBJECTIDENTIFIER • Opaque • TimeTicks • Unsigned32 **Note** The OCTETSTRING can be a string that will be converted to a hexadecimal notation without a trailing NULL, octet string for example, or hexadecimal notation contained in single quotation marks, 'aa:bb:cc' for example.	`.experimental.docsDev.docsDevMIBObjects.docsDevNmAccessTable.docsDevNmAccessEntry.docsDevNmAccessStatus.1, INTEGER, 4`

Encoding	Input	Examples
SrvChangeAct	Unsigned 8-bit integer that is restricted to a range from 0 to 3, or a SrvChangeAct description. The output for the description string is: • 0—Add PHS Rule • 1—Set PHS Rule • 2—Delete PHS Rule • 3—Delete all PHS Rules	`0`

Subtype	One or two comma-separated unsigned integer 8.	`12`
		`12,14`

Topology Mode Encoding	Unsigned 8 bit integer used for subdividing an Operator-delegated IPv6 prefix 1: Favor Depth 2: Favor Width	1 2 Depth Width

Transport address and mask	For IPv4, four-octet IP address in dotted notation followed by the port number, separated by a comma (,).
	For IPv6, in dotted notation or string:
	• Valid IPv6 address in dotted notation followed by the port number, separated by comma (,).
	• A string representation of IPv6 address, followed by the port number, separated by comma (,). For example: *x:x:x:x:x:x:x:x*,1234, where the *x*s are one to four hexadecimal digits of the eight 16-bit pieces of the address.
		IPv4
		`10.112.125.111,5678`
		IPv6
		`2001.db8.0.0.8.800.200c.417a,5678`
		`2001:db8:0:0:8:800:200c:417a,5678`

Unsigned integer 8	0 to 255	`14`
Unsigned integer 16	0 to 65535	`1244`
Unsigned integer 32	0 to 4294967295	`3455335`

Encoding	Input	Examples
Unsigned integer 8 and unsigned integer 16	One unsigned integer 8 and one unsigned integer 16, comma separated.	`3,12324`
Unsigned integer 8 pair	Two unsigned integer 8, comma separated.	`1,3`
Unsigned integer 8 triplet	Three unsigned integer 8, comma separated.	`1,2,3`

Verify	Unsigned 8-bit integer
	To enable verification, the values are:
	• 0
	• Verify
	To disable verification, the values are:
	• 1
	• Don't Verify
	Note The definitions of true and false for the Verify TLV are in line with the DOCSIS 1.1 specification (Option 26.11).
		`0 = verify`
		`1 = don’t verify`

ZTASCII	An ASCII string. The encoded string will be NULL terminated.	`This is an ASCII string`

BITS Value Syntax

When using the BITS type, you must specify either the labels (“interval1 interval2 interval3”) or numeric bit location (“0 1 2”). Note that label values are 1-based and bit values are 0-based.

This is the syntax that uses the bit numbers:

```
option 11 .pktcSigDevR0Cadence.0,STRING,"0 1 2 3 4 5 6 7 8 9 10 11 12 13 14"
```

This is the syntax for the customer octet string (FFFE000000000000) that uses the labels:

```
option 11 .pktcSigDevR0Cadence.0,STRING,"interval1 interval2 interval3
interval4 interval5 interval6 interval7 interval8 interval9 interval10
interval11 interval12 interval13 interval14 interval15"
```

OCTETSTRING Syntax

The OCTETSTRING can be either a string that is converted to hexadecimal notation without a trailing NULL (for example, octet string), or hexadecimal notation contained within single quotation marks (for example, 'aa:bb:cc').

Using Configuration File Utility for Template

You use the configuration file utility to test, validate, and view PacketCable 1.0/1.5/2.0, DOCSIS 1.0/1.1/2.0/3.0/3.1, and CableHome template and configuration files. These activities are critical to successfully deploy individualized configuration files. See Template Files–An Overview, on page 20, for more information on templates.

The configuration file utility is available only when the RDU is installed; the utility is installed in the *BPR_HOME/rdu/bin* directory.

Both the template file being encoded and the binary file being decoded must reside in the directory from which the configuration file utility is invoked.

All examples in this section assume that the RDU is operating and that these conditions apply:

- The Prime Cable Provisioning application is installed in the default home directory (*/opt/CSCObac*).
- The RDU login name is **admin**.
- The RDU login password is **changeme**.

Note Some of the examples in this section were removed whenever the omitted information is of no consequence to the example of its outcome. Instances where this occurs are identified by an ellipses (...) that precedes the example summary.

This section discusses these topics:

- Generating TLV 43s for Multivendor Support, on page 63

Running the Configuration File Utility

In subsequent procedures and examples, the phrase "run the configuration file utility" means to enter the **runCfgUtil.sh** command from the directory specified. To run the configuration file utility, run this command from the *BPR_HOME/rdu/bin* directory:

runCfgUtil.sh *options*

The available *options* include:

- **-c** *shared*—Specifies the CMTS shared secret when parsing a DOCSIS template file. To specify the default shared secret, enter **-c cisco**.
- **-cablehome**—Identifies the input file as a CableHome portal service configuration file. Do not use this with either the **-docsis** or **-pkt** options.
- **-d**—Decodes the binary input file. Do not use this with the **-e** option.
- **-docsis**—Specifies the input file as a DOCSIS configuration file. Do not use this default with the **-pkt** option.
- **-v** *version*—Specifies the DOCSIS version being used. For example, if you are using DOCSIS 1.1, enter **-v 1.1**. If you do not specify the version number, the command defaults to use the latest supported version of DOCSIS. The values that Prime Cable Provisioning supports are 1.0, 1.1, 2.0, 3.0, and 3.1.
- **-e**—Encodes the template input file. Do not use this default with the **-d** option.
- **-g**—Generates a template file from either a DOCSIS, PacketCable, or CableHome binary file.
- **-h** *host:port*—Specifies the host and port. The default port number is 49187.
- **-i** *device-id*—Identifies the device to use when substituting macro variables during template parsing. For example, if the device MAC address is 1,6,00:00:00:00:00:01, enter **-i 1,6,00:00:00:00:00:01**, or if the device DUID is 00:03:00:01:00:18:68:52:75:c0, enter **-i 00:03:00:01:00:18:68:52:75:c0**. When using this option, you must also use the **-u** and **-p** options, respectively, to specify the username and password. Do not use this with the **-m** option.
- **-l** *filename*—Identifies the input file as being on the local file system. For example, if your input file is called *any_file*, enter **-l any_file**. Do not use this with the **-r** option.
- **-loc** locale—Specifies the PacketCable locale such as na, euro, and, ietf (default is na). The default is na. If the MTA is euro-MTA, then the locale should be set to euro.
- **-m** *macros*—Specifies key value pairs for macro variables. The format is key=value. If you require multiple macro variables, use a double comma separator between the key value pairs; for example, key_1=value_1,,key_2=value_2. Do not use this with the **-i** option.
- **-p** *password*—Specifies the password to use when connecting to the RDU. For example, if your password is 123456, enter **-p 123456**.
- **-o** *filename*—Saves a parsed template file as a binary file. For example, if you want the output to be found in a file called *op_file,* enter **-o op_file**.
- **-pkt**—Identifies the input file as a PacketCable MTA configuration file. Do not use this with the **-docsis** option.

- **-r** *filename*—Identifies the input file as a remote file that has been added to the RDU. For example, if your file is called *file25*, enter **-r file25**. When using this option you must also use the **-u** and **-p** options, to specify the username and password, respectively. Do not use this with the **-l** option.
- **-s**—Displays the parsed template or the contents of the binary file in a human-readable format.
- **-t**—Specifies the PacketCable encoding type: **Secure** or **Basic** (the default is Secure).
- **-u** *username*—Specifies the username to use when connecting to the RDU. For example, if your username is admin, enter **-u admin**.
- **-E**—Enables Extended CMTS MIC (EMIC) calculation and identifies the default options for EMIC calculation. The default options are:
  - HMAC type—MMH16
  - EMIC Digest type—Explicit
  - EMIC shared secret as **cisco**.
- **-Ei**—Identifies the EMIC Digest type as implicit for EMIC calculation.
- **-Eh**—Specifies the HMAC type: MD5 or MMH16 (the default is MMH16).
- **-Es** *secret*—Specifies the EMIC shared secret when parsing a DOCSIS template file.

Note The configuration file utility does not include Option 19 (TFTP server timestamp) and Option 20 (TFTP server provisioned modem address) in the template file; the Prime Cable Provisioning TFTP mixing, however, does. Also, options 6 (CM MIC) and 7 (CMTS MIC) are both automatically inserted into the encoded template file. Therefore, you do not have to specify these message integrity checks (MICs).

Adding a Template to Prime Cable Provisioning

To use the configuration file utility to test Prime Cable Provisioning templates:

Step 1 Develop the template as described in Template Files–An Overview, on page 20. If the template includes other templates, make sure all the referenced templates are in the same directory.

Step 2 Run the configuration file utility on the local file system. You can check the syntax for the template, or have the configuration file utility process the template as CRS would, and return output.

If the template contains macro variables, perform these operations in the order specified:

a) Test with command line substitution.
b) Test with a device that has been added to your RDU.

Step 3 Add the template (and any included templates that are used) to the RDU.

Step 4 Run the configuration file utility to parse a file. See Testing Template Processing for an External Template File.

If the template contains macro variables, perform these operations in the order specified:

a) Test with command-line substitution.
b) Test with a device that has been added to your RDU.

Step 5 After all tests succeed, configure a Class of Service to use the template.

Converting a Binary File to a Template File

Use the **runCfgUtil.sh** command to convert binary configuration memory files into template files. Prime Cable Provisioning dynamic configuration generation is based on templates that are created. Automatically converting existing, tested, binary files to template files speeds the process and reduces the possibility of introducing errors.

Syntax Description

runCfgUtil.sh -g -l *binary_file* **-o** *template_file*

- **-g**—Specifies that a template file needs to be generated from an input binary file
- **-l** *binary_file*—Specifies the local input file, including the pathname. In all cases, the input binary filename will have a *.cm* file extension; *bronze.cm* for example.
- **-o** *template_file*—Specifies the output template file, including the pathname. In all cases, the output template file will have a *.tmpl* file extension; for example*, test.tmpl*.

To convert a binary file into a template file:

Step 1 Change directory to */opt/CSCObac/rdu/samples/docsis*.

Step 2 Select a template file to use. This example uses an existing binary file called *unprov.cm*.

Step 3 Run the configuration file utility using this command:

/opt/CSCObac/rdu/bin# **`runCfgUtil.sh -g -l unprov.cm -o test.tmpl -docsis`**

-docsis—Specifies the input file to be a DOCSIS configuration file.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

################################################################
## Template File Generator
## Generated on Fri Oct 12 16:12:51 EST 2007
################################################################

################################################################
## Each generated option will be represented by the following:
## The first line will represent a description of the
## generated option
## The second line will represent the generated option
## The third line will represent the custom version
## of the generated option
################################################################

# (3) Network Access Control
Option 3  01
# Option 3  hex 01

# (4.1) Class ID
Option 4.1 1
# Option 4.1 hex 01
# (4.2) Maximum Downstream Rate
Option 4.2 128000
# Option 4.2 hex 0001F400

# (4.3) Maximum Upstream Rate
Option 4.3 64000
# Option 4.3 hex 0000FA00

# (4.4) Upstream Channel Priority
Option 4.4 1
# Option 4.4 hex 01

# (4.5) Guaranteed Minimum Upstream Channel Data Rate
Option 4.5 0
# Option 4.5 hex 00000000

# (4.6) Maximum Upstream Channel Transmit Burst
Option 4.6 1600
# Option 4.6 hex 0640

# (4.7) Class-of-Service Privacy Enable
Option 4.7 00
# Option 4.7 hex 00

# (11) SNMP MIB Object
Option 11
.iso.org.dod.internet.experimental.docsDev.docsDevMIBObjects.docsDevNmAccessTable.docsDevNmAccessEntry.
docsDevNmAccessStatus.1,INTEGER,createAndGo
# Option 11  hex 3082000F060A2B060103530102010701020104

...

# (18) Maximum Number of CPEs
Option 18  1
# Option 18  hex 01
```

Testing Template Processing for a Local Template File

Use the **runCfgUtil.sh** command to test processing for template files stored on the local file system.

Syntax Description

runCfgUtil.sh -pkt -l *file*

- **-pkt**—Identifies the input file as a PacketCable MTA file.
- **-l**—Specifies that the input file is on the local file system.
- *file*—Identifies the input template file being parsed.

To parse a template file that is on the local file system:

Step 1 Change directory to */opt/CSCObac/rdu/samples/packet_cable*.

Step 2 Select a template file to use. This example uses an existing template file called *unprov_packet_cable.tmpl*. The **-pkt** option is used because this is a PacketCable MTA template.

Step 3 Run the configuration file utility using this command:

/opt/CSCObac/rdu/bin/**`runCfgUtil.sh -pkt -l unprov_packet_cable.tmpl`**

unprov_packet_cable.tmpl—Identifies the input template file being parsed.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                File Bytes         Option             Description        Value


0                  FE0101             254                Telephony Config   1
                                                         File Start/End


3                  0B153013060E       11                 SNMP MIB Object    .iso.org.dod.internet
                    2B06010401A                                             .private.enterprises
                   30B0202010101                                            .cableLabs.clabProject
                    0700020102                                              .clabProjPacketCable
                                                                            .pktcMtaMib.pktcMtaMibObjects

                                                                            .pktcMtaDevBase.
                                                                            pktcMtaDevEnabled.0,INTEGER,false(2)


...

0 error(s), 0 warning(s) detected. Parsing of unprov_packet_cable.tmpl was successful.
The file unprov_packet_cable.tmpl was parsed successfully in 434 ms.
The parser initialization time was 92 ms.
The parser parse time was 342 ms.
```

Testing Template Processing for an External Template File

Use the **runCfgUtil.sh** command to test processing of external template files.

Syntax Description

runCfgUtil.sh -docsis -r *file* **-u** *username* **-p** *password*

- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the input template file being parsed.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.
- **-docsis**—Identifies the file as a DOCSIS template.

To parse a template file that has been added to the RDU:

Step 1 Select a template file to use. This example uses an existing template file called *unprov.tmpl*. The **-docsis** option is used because a DOCSIS template is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin#  runCfgUtil.sh -docsis -r unprov.tmpl -u admin -p changeme
```

- **unprov.tmpl**—Identifies the input file.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

After running the utility, results similar to these should appear:

Note The results shown here are for illustration only and have been trimmed for brevity.

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.1

Off                File Bytes        Option            Description       Value


0                  030101            3                 Network Access    On
                                                       Control


3                  041F              4                 Class of Service


5                  010101            4.1               Class ID          1


8                  02040000FA00      4.2               Maximum           128000 bits/sec
                                                       Downstream Rate


14                 03040000FA00      4.3               Maximum Upstream  64000 bits/sec
                                                       Rate


20                 040101            4.4               Upstream Channel  1
                                                       Priority


...


252                06108506547F      6                 CM MIC            8506547FC9152B44
                   C9152B44DB95                        Configuration     DB955420843EF6FE
                   5420843EF6FE                        Setting


270                0710644B675B      7                 CMTS MIC          644B675B70B7BD3E
                   70B7BD3E09AC                        Configuration     09AC210F794A1E8F
                   210F794A1E8F                        Setting


288                FF                255               End-of-Data
                                                       Marker


289                00                0                 PAD


290                00                0                 PAD


291                00                0                 PAD

0 error(s), 0 warning(s) detected. Parsing of unprov.tmpl was successful.
The file unprov.tmpl was parsed successfully in 375 ms.
The parser initialization time was 63 ms.
The parser parse time was 312 ms.
```

Testing Template Processing for a Local Template File and Adding Shared Secret

Use the **runCfgUtil.sh** command to test processing for a template file and add a shared secret that you specify.

Syntax Description

runCfgUtil.sh -e -docsis -l *file* **-c** *shared*

- **-e**—Identifies the encode option.
- **-docsis**—Identifies the input file as a DOCSIS template file.
- **-l**—Specifies that the input file is on the local file system.
- *file*—Identifies the input template file being parsed.
- **-c**—Specifies the CMTS shared secret when parsing a DOCSIS template file.
- *shared*—Identifies the new shared secret. The default shared secret is **cisco**.

To parse a locally saved template file, and set a user-specified shared secret:

Step 1 Change directory to */opt/CSCObac/rdu/templates*.

Step 2 Select a template file to parse. This example uses an existing template file called *unprov.tmpl*. The **-docsis** option is used because this is a DOCSIS template.

Step 3 Run the configuration file utility using this command:

/opt/CSCObac/rdu/bin/**`runCfgUtil.sh -e -docsis -l unprov.tmpl -c shared`**

- **unprov.tmpl**—Identifies the input file on the local file system.
- **shared**—Identifies that new shared secret.

After running the utility, results similar to these should appear:

Cisco Prime Cable Provisioning Configuration Utility Version: 5.1				
Off	File Bytes	Option	Description	Value
0	030100	3	Network Access Control	Off
3	041F	4	Class of Service	
5	010101	4.1	Class ID	1
8	02040001F400	4.2	Maximum Downstream Rate	128000 bits/sec

14	03040000FA00	4.3	Maximum Upstream Rate	64000 bits/sec
20	040101	4.4	Upstream Channel Priority	1
...				
252	06108506547F C9152B44DB95 5420843EF6FE	6	CM MIC Configuration Setting	8506547FC9152B44 DB955420843EF6FE
270	0710644B675B 70B7BD3E09AC 210F794A1E8F	7	CMTS MIC Configuration Setting	644B675B70B7BD3E 09AC210F794A1E8F
288	FF	255	End-of-Data Marker	
289	00	0	PAD	
290	00	0	PAD	
291	00	0	PAD	

```
0 error(s), 0 warning(s) detected. Parsing of unprov.tmpl was successful.
The file unprov.tmpl was parsed successfully in 375 ms.
The parser initialization time was 63 ms.
The parser parse time was 312 ms.
```

Testing Template Processing for a Local Template File and Adding EMIC Shared Secret

Use the **runCfgUtil.sh** command to test the processing for a template file and add a EMIC shared secret that you specify.

Example 1:

This example describes how you can use **runCfgUtil**.sh command to enable EMIC, and set:

- MMH16 as the HMAC type.
- EMIC Digest Explicit option.
- cisco as the EMIC shared secret for EMIC calculation.

Syntax Description

runCfgUtil.sh -E -docsis -l *filename*

- **-E**—Enables EMIC calculation.

- **-docsis**—Identifies the input file as a DOCSIS template file.
- **-l** *filename*—Specifies the input template file, including the pathname. In all cases, the input template file will have a .tmpl file extension; for example, test.tmpl.

To perform EMIC calculation with default settings:

Step 1 Select a template file to use. This example uses an existing template file called *unprov.tmpl*. The **-docsis** option is used because a DOCSIS template is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin/runCfgUtil.sh -E -l test.tmpl
```

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.1
```

Off.	File bytes.	Option.	Description.	Value.
0	030101	3	Network Access Control	On
3	041F	4	Class of Service	
5	010101	4.1	Class ID	1
8	0204001F4000	4.2	Maximum Downstream Rate	2048000 bits/sec
14	03040007D000	4.3	Maximum Upstream Rate	512000 bits/sec
20	040106	4.4	Upstream Channel Priority	6
23	050400000000	4.5	Guaranteed Minimum Upstream Channel Data Rate	0 bits/sec
29	06020640	4.6	Maximum Upstream Channel Transmit Burst	1600 bytes
33	070100	4.7	Class-of-Service Privacy Enable	Disabled
249	120103	18	Maximum Number of CPEs	3

252	2B1C	43	DOCSIS Extension Field	
254	0803FFFFFF	43.8	Vendor ID	FF-FF-FF
259	0615	43.6	Extended CMTS MIC Configuration Setting	
261	010102	43.6.1	Extended CMTS MIC HMAC type	2
264	020678007BEC 1C80	43.6.2	Extended CMTS MIC Bitmap	78007BEC1C80
272	03081605487D 3D7A9403	43.6.3	Explicit Extended CMTS MIC Digest Subtype	1605487D3D7A9403
282	06108BFC8016 39BE8D7F396E E49D402832FC	6	CM MIC Configuration Setting	8BFC801639BE8D7F 396EE49D402832FC
300	071053F94674 11C355EF01D0 104995AB9797	7	CMTS MIC Configuration Setting	53F9467411C355EF 01D0104995AB9797
318	FF	255	End-of-Data Marker	
319	00	0	PAD	

Example 2:

This example describes how you can use **runCfgUtil.sh** command to enable EMIC and to change the default settings:

Syntax Description

runCfgUtil.sh -E -Ei -Eh *MD5* **-Es** *secret* **-l** *filename*

- **-E**—Enables EMIC calculation.
- **-Ei**—Specifies EMIC Digest type as implicit for EMIC calculation.
- **-l** *filename*—Specifies the input template file, including the pathname. In all cases, the input template file will have a .tmpl file extension; for example, test.tmpl.
- **-Eh**—Specifies the HMAC type: MD5 or MMH16 (the default is MMH16).
- **-Es** *secret*—Specifies the EMIC shared secret when parsing a DOCSIS template file.

To perform EMIC calculation using the options which are not included as defaults:

Step 1 Select a template file to use. This example uses an existing template file called *unprov.tmpl*. The **-docsis** option is used because a DOCSIS template is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin/runCfgUtil.sh -E -Ei -Eh MD5 -Es secret -l test.tmpl
```

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.1
```

Off.	File bytes.	Option.	Description.	Value.
0	030101	3	Network Access Control	On
3	041F	4	Class of Service	
5	010101	4.1	Class ID	1
8	0204001F4000	4.2	Maximum Downstream Rate	2048000 bits/sec
14	03040007D000	4.3	Maximum Upstream Rate	512000 bits/sec
20	040106	4.4	Upstream Channel Priority	6
23	050400000000	4.5	Guaranteed Minimum Upstream ChannelData Rate	0 bits/sec
29	06020640	4.6	Maximum Upstream Channel Transmit Burst	1600 bytes
33	070100	4.7	Class-of-Service Privacy Enable	Disabled
249	120103	18	Maximum Number of CPEs	3
252	2B12	43	DOCSIS Extension Field	
254	0803FFFFFF	43.8	Vendor ID	FF-FF-FF

259	060B	43.6	Extended CMTS MIC Configuration Setting	
261	010101	43.6.1	Extended CMTS MIC HMAC type	1
264	020678007BEC 1C80	43.6.2	Extended CMTS MIC Bitmap	78007BEC1C80
272	06107E6CF875 32C551791CCF 34770C94FA03	6	CM MIC Configuration Setting	7E6CF87532C55179 1CCF34770C94FA03
290	0710C9F8007A 40E4913779D3 C1C53599141B	7	CMTS MIC Configuration Setting	C9F8007A40E49137 79D3C1C53599141B
308	FF	255	End-of-Data Marker	
309	00	0	PAD	
310	00	0	PAD	
311	00	0	PAD	

Specifying Macro Variables at the Command Line

Use the **runCfgUtil.sh** command to specify macro variables.

Syntax Description

runCfgUtil.sh -e -l *file* **-m** *“macros”*

- **-e**—Identifies the encode option.
- **-l**—Specifies the input file is on the local file system.
- *file*—Identifies the input template file being parsed.
- **-m**—Specifies the macro variables to be substituted when parsing a template.
- *“macros”*—Identifies the desired macros. When multiple macro variables are required, insert a double comma separator between each macro.

To specify values for macro variables at the command line:

Step 1 Change directory to */opt/CSCObac/rdu/templates*.

Step 2 Select a template file to use.

Step 3 Identify the macro variables in the template. In this example, the macro variables are macro1 (option 3) and macro11 (option 4.2).

Step 4 Identify the values for the macro variables. The value for macro1 will be set to 1, and the value for macro11 to 64000.

Step 5 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -e -l macro.tmpl -m "macro1=1,,macro11=64000"
```

- **macro.tmpl**—Identifies the input file.
- **macro1=1,macro11=64000**—Identifies the key value pairs for macro variables. Because multiple macro variables are necessary, a double comma separator is inserted between the key value pairs.

  After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                File Bytes         Option             Description        Value


0                  030101             3                  Network Access     On
                                                         Control


3                  041F               4                  Class of Service


5                  010101             4.1                Class ID           1


8                  02040000FA00       4.2                Maximum Downstream 64000 bits/sec
                                                          Rate


14                 03040000FA00       4.3                Maximum Upstream   64000 bits/sec
                                                         Rate


20                 040101             4.4                Upstream Channel   1
                                                         Priority


...

0 error(s), 0 warning(s) detected. Parsing of macro.tmpl was successful.
The file macro.tmpl was parsed successfully in 854 ms.
The parser initialization time was 76 ms.
The parser parse time was 778 ms.
```

Specifying a Device for Macro Variables

Use the **runCfgUtil.sh** command to specify a device for macro variables.

Syntax Description

runCfgUtil.sh -e -r *file* **-i** *MAC* **-u** *username* **-p** *password*

- **-e**—Identifies the encode option.
- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the input template file being parsed.
- **-i**—Specifies the device to use when parsing macro variables.
- *MAC*—Identifies the MAC address of the device.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.

To specify a device to be used for macro variable substitution:

Step 1 Select a template file to use. This example uses the existing template file, *macro.tmpl*.

Step 2 Identify the macro variables in the template. In this example, the macro variables are macro1 (option 3) and macro11 (option 4.2).

Step 3 Identify the device to use. This example assumes that the device exists in the RDU and has the macro variables set as properties. The value for macro1 will be set to 1, and the value for macro11 to 64000.

Step 4 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin/runCfgUtil.sh -e -r macro.tmpl -i "1,6,00:01:02:03:04:05" -u admin -p changeme
```

- **macro.tmpl**—Identifies the input file.
- **1,6,00:01:02:03:04:05**—Identifies the MAC address of the device. The MAC address used here is for example purposes only.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                 File Bytes          Option              Description       Value


0                   030101              3                  Network Access     On
                                                           Control


3                   041F                4                  Class of Service


5                   010101              4.1                Class ID           1


8                   02040000FA00        4.2                Maximum Downstream 64000 bits/sec
                                                            Rate


14                  03040000FA00        4.3                Maximum Upstream   64000 bits/sec
                                                           Rate
```

```
20              040101          4.4             Upstream Channel   1
                                                Priority

...

0 error(s), 0 warning(s) detected. Parsing of macro.tmpl was successful.
The file macro.tmpl was parsed successfully in 159 ms.
The parser initialization time was 42 ms.
The parser parse time was 117 ms.
```

Specifying Output to a Binary File

Use the **runCfgUtil.sh** command to specify the output of a parsed template as a binary file.

Syntax Description

runCfgUtil.sh -l *input_file* **-o** *output_file*

- **-l**—Specifies that the input file is on the local file system.
- *input_file*—Identifies the input template file being parsed.
- **-o**—Specifies that the parsed template file is to be saved as a binary file.
- *output_file*—Identifies the name of the file in which the binary contents of the parsed template file are stored.

To specify the output from parsing a template to a binary file:

Step 1 Change directory to */opt/CSCObac/rdu/templates*.

Step 2 Select a template file to use.

Step 3 Identify the name of the output file. This example uses *unprov.cm*.

Step 4 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -l unprov.tmpl -o unprov.cm
```

- **unprov.tmpl**—Identifies the existing template file being parsed into a binary file.
- **unprov.cm**—Identifies the output filename to be used.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0
```

```
0 error(s), 0 warning(s) detected. Parsing of unprov.tmpl was successful.
The file unprov.tmpl was parsed successfully in 595 ms.
The parser initialization time was 262 ms.
The parser parse time was 333 ms.
```

Viewing a Local Binary File

Use the **runCfgUtil.sh** command to view a binary file stored in the local system.

Syntax Description

runCfgUtil.sh -d -l *file*

- **-d**—Specifies that the command is going to decode a binary input file for viewing.
- **-l**—Identifies that the input file resides on the local file system.
- *file*—Identifies the existing binary input file to be viewed.

To view a binary file that is on the local file system:

Step 1 Change directory to */opt/CSCObac/rdu/samples/packet_cable*.

Step 2 Select a binary file to view.

Step 3 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -d -l unprov_packet_cable.bin
```

unprov_packet_cable.bin—Identifies the existing binary input file to be viewed.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0
Warning: Expecting config file of type docsis, but input file is of type pktc1.0.  Decoding as pktc1.0


Off                 File Bytes          Option              Description         Value


0                   FE0101              254                 Telephony Config    1
                                                            File Start/End


      3             0B153013060E        11                  SNMP MIB Object     .iso.org.dod.internet.private.enterpri
                    2B06010401A3                                                ses.cableLabs.clabProject.clabProjPack
                    0B0202010101                                                etCable.pktcMtaMib.pktcMtaMibObjects.p
                    0700020102                                                  ktcMtaDevBase.pktcMtaDevEnabled.0,INTE
                                                                                GER,fals e(2)


...
```

Note The warning in this example appears because the default input file is DOCSIS, and this example uses a binary PacketCable file. If you use the **-pkt** option to specify the input file as a PacketCable file, the warning does not appear. For example:

```
/opt/CSCObac/rdu/bin/# runCfgUtil.sh -d -pkt -l unprov_packet_cable.bin
```

Viewing an External Binary File

Use the **runCfgUtil.sh** command to view an external binary file.

Syntax Description

runCfgUtil.sh -d -r *file* **-u** *username* **-p** *password*

- **-d**—Specifies that the command is going to decode a binary input file for viewing.
- **-r**—Identifies the input file as a file that has been added to the RDU.
- *file*—Identifies the existing binary file in the RDU.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.

To view a binary file that has been added to the RDU:

Step 1 Select a binary file to view. This example uses the existing binary file *unprov.cm*, and assumes that the RDU is localhost:49187.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -d -r unprov.cm -u admin -p changeme
```

- **unprov.cm**—Identifies the existing binary file in the RDU.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                File Bytes         Option              Description       Value


0                  030100             3                  Network Access     Off
                                                         Control


3                  041F               4                  Class of Service


5                  010101             4.1                Class ID           1
```

```
8                   02040001F400       4.2                Maximum Downstream 128000 bits/sec
                                                           Rate


14                  03040000FA00       4.3                Maximum Upstream   64000 bits/sec
                                                          Rate


20                  040101             4.4                Upstream Channel   1
                                                          Priority


...


252                 06108506547F       6                  CM MIC             8506547FC9152B44
                    C9152B44DB95                          Configuration      DB955420843EF6FE
                    5420843EF6FE                          Setting


270                 0710644B675B       7                  CMTS MIC           644B675B70B7BD3E
                    70B7BD3E09AC                          Configuration      09AC210F794A1E8F
                    210F794A1E8F                          Setting


288                 FF                 255                End-of-Data Marker


289                 00                 0                  PAD


290                 00                 0                  PAD


291                 00                 0                  PAD

0 error(s), 0 warning(s) detected. Parsing of unprov.tmpl was successful.
The file unprov.tmpl was parsed successfully in 375 ms.
The parser initialization time was 63 ms.
The parser parse time was 312 ms.
```

Activating PacketCable Basic Flow

Use the **runCfgUtil.sh** command to support the generation and insertion of the PacketCable Basic Flow integrity hash into a Basic Flow static configuration file.

Syntax Description

runCfgUtil.sh -t {**basic** | **secure**} **-pkt -r** *filename* **-u** *username* **-p** *password*

- **basic**—Calculates and inserts a PacketCable Basic Flow integrity hash into an MTA static configuration file.
- **secure**—Stops the insertion of the PacketCable Basic Flow integrity hash into an MTA static configuration file. This is the default setting.
- **-r**—Identifies the input file as a file that has been added to the RDU.

- *filename*—Identifies the input file.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*—Specifies the password to use when connecting to the RDU.
- **-pkt**—Identifies the input file as a PacketCable MTA configuration file.

To support the generation and insertion of the PacketCable Basic Flow integrity hash into a Basic flow static configuration file:

Step 1 Select the Basic Flow static configuration file into which you want to insert the PacketCable Basic Flow integrity hash. This example uses the *example_mta_config.tmpl*.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -t basic -pkt -r example_mta_config.tmpl -u admin -p changeme
```

- **example_mta_config.tmpl**—Identifies the Basic Flow static configuration file.
- **admin**—Identifies the default username.
- **changeme**—Identifies the default password.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                File Bytes         Option             Description        Value

0                  FE0101             254                Telephony Config   1
                                                         File Start/End

                                                                            .iso.org.dod.internet.
3                  0B153013060E       11                 SNMP MIB Object    private.enterprises.ca
                   2B06010401A3                                             bleLabs.clabProject.cl
                   0B0202010101                                             abProjPacketCable.pktc
                   0700020101                                               MtaMib.pktcMtaMibObjec
                                                                            ts.pktcMtaDevBase.pktc
                                                                            MtaDevEnabled.0,INTEGE
                                                                            R,true(1)

                                                                            .iso.org.dod.internet.
26                 0B2530230610       11                 SNMP MIB Object    private.enterprises.ca
                   2B06010401A3                                             bleLabs.clabProject.cl
                   0B0202020102                                             abProjPacketCable.pktc
                   01010109040F                                             SigMib.pktcSigMibObjec
                   434D532E4950                                             ts.pktcNcsEndPntConfig
                   464F4E49582E                                             Objects.pktcNcsEndPntC
                   434F4D                                                   onfigTable.pktcNcsEndP
                                                                            ntConfigEntry.pktcNcsE
                                                                            ndPntConfigCallAgentId
                                                                            .9,STRING,CMS.IPFONIX.
                                                                            COM

...
```

```
371              FE01FF           254              Telephony Config   255
                                                   File Start/End

0 error(s), 0 warning(s) detected. Parsing of example_mta_config.tmpl was successful.
The file example_mta_config.tmpl was parsed successfully in 100 ms.
The parser initialization time was 44 ms.
The parser parse time was 56 ms.
```

A file with a *.tmpl* extension is assumed to be a dynamic configuration template, for which the Basic hash calculation and insertion occur transparently during template processing; as a result, you can use the same template for provisioning in the Secure and Basic modes.

However, if you want to convert a Secure static binary configuration file to a Basic static configuration file before inserting the hash, follow this procedure:

a) Convert the Secure static file to a template, by using:

   # **runCfgUtil -l** *input_static_filename* **-pkt -g -o** *output_template_filename*

b) Convert the Secure static template into a Basic static configuration file, by using:

   # **runCfgUtil -t basic -l** *input_template_name* **-pkt -o** *output_Basic_static_filename*

   This command calculates and inserts the Basic integrity hash into the Basic static configuration file.

Generating TLV 43s for Multivendor Support

Use the **runCfgUtil.sh** command to generate TLV 43s in order to provide multivendor support.

Syntax Description

runCfgUtil.sh -docsis -r *filename* **-u** *username* **-u** *password*

- **-docsis**—Identifies the input file as a DOCSIS template file.
- *filename*—Identifies the input template file being parsed.
- **-r**—Identifies the input file as a file that has been added to the RDU.
- **-u** *username*—Specifies the username to use when connecting to the RDU.
- **-p** *password*— Specifies the password to use when connecting to the RDU.

To generate TLV 43s using a template file that has been added to the RDU:

Step 1 Select a template file to use. This example uses an existing template file called *test.tmpl*. The **-docsis** option is used because a DOCSIS template is being used.

Step 2 Run the configuration file utility using this command:

```
/opt/CSCObac/rdu/bin# runCfgUtil.sh -docsis -r test.tmpl -u admin -p changeme
```

- **test.tmpl**—Identifies the DOCSIS configuration file.
- **admin**—Identifies the default username.

- **changeme**—Identifies the default password.

After running the utility, results similar to these should appear:

```
Cisco Prime Cable Provisioning Configuration Utility
Version: 5.0

Off                 File Bytes          Option              Description         Value


0                   2B14                43                  DOCSIS Extension
                                                            Field


2                   0803FFFFFF          43.8                Vendor ID           FF-FF-FF


7                   050D                43.5                L2VPN Encoding


9                   010502345600        43.5.1              VPNID Subtype       0234560003
                    03


16                  0204                43.5.2              NSI Encapsulation
                                                            Subtype


18                  02020019            43.5.2.2            IEEE 802.1Q Format  25
                                                            Subtype


22                  2B0B                43                  DOCSIS Extension
                                                            Field


24                  080300000C          43.8                Vendor ID           00-00-0C (CISCO
                                                                                SYSTEMS, INC.)


29                  010418017A30        43.1                Static Downstream   402750000
                                                            Frequency


35                  2B1D                43                  DOCSIS Extension
                                                            Field


37                  080300000C          43.8                Vendor ID           00-00-0C (CISCO
                                                                                SYSTEMS, INC.)


42                  0316626F6F74        43.3                Update Boot Monitor boot_monitor_image.bin
                    5F6D6F6E6974                             Image
                    6F725F696D61
                    67652E62696E


66                  061071E79068        6                   CM MIC Configuration 71E790683DE8B995
                    3DE8B9950536                             Setting            05368936F4C5312F
                    8936F4C5312F
```

```
84                  0710DB0EED14       7                   CMTS MIC           DB0EED14B5B3428D
                    B5B3428D2B15                           Configuration      2B150DA582B41A54
                    0DA582B41A54                           Setting


102                 FF                 255                 End-of-Data Marker


103                 00                 0                   PAD


0 error(s), 0 warning(s) detected. Parsing of test.tmpl was successful.
The file test.tmpl was parsed successfully in 250 ms.
The parser initialization time was 109 ms.
The parser parse time was 141 ms.
```

Using MIBs with Dynamic DOCSIS Templates

For a full list of MIBs that Prime Cable Provisioning ships with, see SNMP VarBind.

You can add MIBs using an application programming interface (API) call or by modifying *rdu.properties*. For more details, see Configuring PacketCable

You can add SNMP TLVs to a template:

- When no MIB is available. See Adding SNMP TLVs Without a MIB.
- With vendor-specific MIBs. See Adding SNMP TLVs With Vendor-Specific MIBs.

MIB Management Enhancements

Following list is supported from Prime Cable Provisioning 5.2.1 for the MIB management:

1. MIB enhancements provide more management options for the MIB files used by the RDU server and the RDU server will load the MIB files available in the RDU database. In earlier releases, the MIB files were loaded from the file system (BPR_HOME/rdu/mibs directory).
2. In the Administrative Web UI, under Configuration > Defaults > System Defaults menu option, for the MIB List field, a new button (Select MIB Files) is added to configure the MIB files from the displayed list that are available in the database instead of entering the file names manually in text box.
3. In the Administrative Web UI, under Configuration > Files page, for MIB file type, a new quick view button is added to view the MIB file details (File Name, Module Name, Parent Modules, Referenced-by Modules).
4. A new option -mib is added in runCfgUtil tool to validate and list the dependencies of MIBs. By default it will validate all the files available in BPR_HOME/rdu/mibs directory. If a filename/directory name is provided as argument, then it will validate and list the dependencies for that file/files available in that directory.
5. The validations that are added in the API for the MIB management are explained in the following table:

Table 4: MIB Management API Validations

API	Task	Sample Error Message
AddFile	Adding an invalid MIB file, <arris_cm.mib>.	Failed to add MIB file <arris_cm.mib>. Not a valid MIB file: <arris_cm.mib>.
	Adding a MIB file <arris_cm.mib> without adding the parent MIB <ARRIS-MIB>.	Failed to add MIB file <arris_cm.mib>. Cannot find the dependent MIB file <ARRIS-MIB> in database.
	Adding a new MIB file with a name that already exists in the database, <SNMPv2-SMI>.	Failed to add MIB file <SNMPv2-SMI>. File already exist in the database.
	Adding a MIB file which is already available in database <URI-TC-MIB>. Adding the same MIB file with a different name <Renamed_URI-TC-MIB>.	Failed to add MIB file <Renamed_URI-TC-MIB.MIB> module <URI-TC-MIB> already exist in database with file name <uri-tc-mib>.
DeleteFile	Delete a MIB file <ARRIS-MIB> which is a parent for some other MIBs.	Failed to delete MIB file <arris-mib>. MIB file(s) <arris_cm.mib, arris_mta_pp.mib> are dependent on MIB file <arris-mib>.
	Delete a MIB file which is loaded. (configured in System Defaults MIB LIST).	Failed to delete MIB file <arris-mib>. The MIB file is currently in use. Requires System Default configuration to be updated under administrator's guidance.
ReplaceFile	Replace a MIB file <ARRIS-MIB> with invalid content (raw text file).	Replace operation failed for MIB file <arris-mib>. Not a valid MIB file: <ARRIS-MIB>.
	Replace a MIB file <ARRIS-MIB> with another valid MIB file <arris_cm.mib>.	Replace operation failed for MIB file <arris-mib.MIB>. MIB module <ARRIS-CM-DEVICE-MIB> conflicts with existing MIB module <ARRIS-MIB>.
	Replace a MIB file <arris_mta_pp.mib> without adding the new parent MIB <ARRIS-ROUTER-DEVICE-MIB>.	Unable to replace the MIB <arris_mta_pp.mib>. Cannot find the dependent MIB module <ARRIS-ROUTER-DEVICE-MIB> in database.
	Replace a MIB file <ARRIS-MIB> by removing few OIDs which are being used by its children.	Replace operation failed for MIB file <arris-mib.MIB>. MIB file <arris-mib.MIB> is not compatible with child MIB(s).

Note The runCfgUtil script will continue to use the MIBs from the file system (BPR_HOME/rdu/mibs) and not from the RDU Database.

MIB Migration

In Prime Cable Provisioning 6.1.1 migration, migrateDB script does not migrate all the custom files from MIBs directory. It migrates only the MIB file names mentioned in the rdu.properties. If the mibList property is not defined in rdu.properties, then it uses the MIB file names from the System Default Configuration and migrates those files to the database.

Migrating User-Defined MIBs

The Prime Cable Provisioning database migration procedure requires that you migrate the components in the sequence recommended in below-mentioned sections.

Note For steps 1-4, see Cisco Prime Cable Provisioning 6.1.1 Quick Start Guide

Step 1: Backup the RDU database.

Step 2: Recover the backed up RDU database.

Step 3: Verify the database integrity.

Step 4: Backup the property files.

Step 5: Migrate the RDU database along with the customer specific MIB files. Follow the steps provided in Cisco Prime Cable Provisioning 6.1.1 Quick Start Guide along with -mibdir option explained below:

- If you have added any custom MIB files in your current installation (5.1 or 5.2), you will need to first back them up from $BPR_HOME/rdu/mibs/ and then copy them manually to the Prime Cable Provisioning 6.1.1 server.

  Note The backed up MIB files should be copied to a location in Prime Cable Provisioning 6.1.1 server other than default $BPR_HOME/rdu/mibs, to avoid overwriting default 6.1.1 MIB files.

- Run the migrateDb.sh tool on the backed up database and backed up custom MIB directory if present. The migrateDb.sh script resides in the $BPR_HOME/migration directory. The migrateDb.sh script supports a new MIB migrating option as shown below:

-mibdir: An optional parameter with custom MIB directory path. -mibdir option should be followed with a directory path which exists in file system and contains custom MIB files to be loaded.

For Example:

```
# $BPR_HOME/migration/migrateDb.sh -dbdir /var/backup/rdu-backup-20120829-031028 -mibdir /tmp/mibs
```

-dbdir: Specifies the location of the database backup that is to be migrated; in this case, /var/backup.

-mibdir: Specifies the location of the custom mibs backup that is to be migrated; in this case, /tmp/mibs.

Note The MIB migration happens along with the database migration and it cannot be performed separately.