



APPENDIX **A**

Using Cisco CMTS MIBs

This chapter describes the objects and MIBs that are needed to use Simple Network Management Protocol (SNMP) requests to perform the tasks on a Cisco CMTS universal broadband router. This chapter contains the following sections:

- [Tips and Guidelines, page A-1](#)
- [Obtaining Basic Information About the Router, page A-3](#)
- [Managing Physical Components, page A-4](#)
- [Generating SNMP Traps, page A-12](#)
- [Monitoring SYSLOG Messages, page A-15](#)
- [Displaying Information About Cable Modems, page A-17](#)
- [Monitoring Spectrum Management, page A-20](#)
- [Using Flap Lists, page A-29](#)
- [Using Subscriber Traffic Management, page A-34](#)
- [Usage-Based Billing, page A-36](#)
- [Identifying Cisco Unique Device Identifiers, page A-40](#)
- [DOCS-DSG-IF-MIB Validation Requirements, page A-41](#)

Tips and Guidelines

When using SNMP to manage the Cisco CMTS router, be aware of the following points.

IF-MIB Caching

In Cisco IOS Release 12.2(15)BC1 and later releases, the Cisco CMTS routers implemented a cache to allow continuous polling of the ifTable interface counters, without creating spikes in the CPU usage. An SNMP request for these counters returns the values that were last stored in the counter cache memory, instead of returning the current run-time value of these counters. This improves performance as it avoids polling each line card to obtain these counters when an SNMP request is made.

The ifTable counter cache is updated approximately every 10 seconds, which means that if you read the ifTable interface counters more quickly than every 10 seconds, the SNMP request might not return new values. The run-time counters do continue to increment, however, to account for the actual traffic occurring on the interfaces, and another SNMP request in 10 seconds does show the new values.

SNMP-Based and CLI-Based Counters

The SNMP specifications do not allow most SNMP-based counters to be cleared, except at system initialization. Instead, during normal operations the counters continue incrementing until they reach their maximum value, at which point they wrap around to zero and continue incrementing again.

This behavior requires the following considerations when managing the router using SNMP commands:

- 32-bit counters—A 32-bit counter wraps around to zero after reaching approximately 4.2 billion. On a busy router, this means that byte and packet counters could wrap around after only a few days. To ensure that you are maintaining the correct counts for packets and other objects, regularly poll the desired counters and always save the previous values. Subtract the previous value from the current value, and if the difference between the two counters becomes negative, it indicates that the counters have wrapped.

To accurately total the counters over a period of several weeks or months, you might also need to keep track of the number of times that the counter wraps during this time period. You should poll the counters often enough so that they do not wrap around to zero more than once without being detected.



Tip

Some SNMPv3 MIBs are beginning to include 64-bit counters, as well as 32-bit counters, for many of the same objects. If given a choice, use the 64-bit counters, as they do not wrap around to zero for months or years.

- Counting from a specified event or time period—SNMP-based counters begin incrementing from zero when the router is powered on, and continue incrementing until they wrap. To track the number of packets or other objects from a particular event, you must save the value of the counters at the time of the event. Then when you want to obtain a new packet count, compare the current value of the counters with the saved value.
- Comparison with command-line interpreter (CLI) values—Many **show** commands have a corresponding **clear** command that resets the counters to zero. The **clear** command, however, affects only the counters that are displayed by the CLI, not the SNMP-based counters. In addition, many CLI-based counters automatically reset whenever a certain function, such as resetting an interface, is performed. This means that the counters displayed using CLI commands are not usually the same as the counters displayed by SNMP commands. Be aware of these differences when comparing the CLI-based and SNMP-based counters.

Redundant PRE Modules on the Cisco uBR10012 Router

On a Cisco uBR10012 router running Cisco IOS Release 12.2BC, SNMP configuration commands are not synchronized to the standby redundant Performance Routing Engine (PRE) module, as is the case with CLI commands. This means that any configuration that is done with SNMP commands is not active when a switchover occurs and the standby PRE module becomes active. When the router switches back to the original PRE module, the original configuration is restored.

For critical configurations, use CLI commands and save them to the startup-config to ensure that they are active during any switchovers.

Obtaining Basic Information About the Router

Basic information about the Cisco CMTS router can be obtained from objects in the following MIBs:

- [OLD-CISCO-CHASSIS-MIB, page A-3](#)
- [SNMPv2-MIB, page A-3](#)
- [ENTITY-MIB, page A-4](#)

OLD-CISCO-CHASSIS-MIB

The following object in the OLD-CISCO-CHASSIS-MIB provides a convenient location to store the chassis serial number for the router, so that it can be easily retrieved when calling Cisco Technical Support:

- **chassisId**—Provides the serial number or ID number for the chassis, as defined by the **snmp-server chassis-id** command, which is typically used to identify the service contract and levels of service that you have purchased from Cisco Technical Support. This object defaults to the empty string, so you must use the **snmp-server chassis-id** command to set the value of this object before you can retrieve it.

```
csh% getmany -v2c 10.10.11.12 public chassisId  
  
chassisId.0 = TBA06500113
```

SNMPv2-MIB

The following objects in the SNMPv2-MIB provide basic information about the router, its software, and other run-time information:

- **sysDescr**—Provides an overall description of the router, including its model number and the version of Cisco IOS software that it is running. For example:

```
csh% getmany -v2c 10.10.11.12 public sysDescr  
  
sysDescr.0 = Cisco Internetwork Operating System Software  
IOS (tm) 10000 Software (UBR10K-K8P6-M), Released Version 12.2(15)BC1  
Copyright (c) 1986-2004 by cisco Systems, Inc.  
Compiled Fri 23-Jan-04 23:56 by atifg
```

- **sysObjectID**—Provides the specific model number, as it is defined in the CISCO-PRODUCTS-MIB. For example:

```
csh% getmany -v2c 10.10.11.12 public sysObjectID  
  
sysObjectID.0 = ciscoProducts.ciscoUBR10012
```

- **sysName**—Provides the host name for the router, as assigned by the **hostname** command. For example:

```
csh% getmany -v2c 10.10.11.12 public sysName  
  
sysName.0 = UBR10012-Router
```

- **sysUpTime**—Provides the time, in hundredths of a second, since the router was last initialized. For example:

```
csh% getmany -v2c 10.10.11.12 public sysUpTime
```

```
sysUpTime.0 = 138389875
```

- **sysContact**—Provides the name, phone number, or other identifying information for the person or department responsible for this router, as it was entered using the **snmp-server contact** command. For example:

```
csh% getmany -v2c 10.10.11.12 public sysContact
```

```
sysContact.0 = IT Support at 408-555-1212 or epage it-support
```

- **sysLocation**—Provides a description of the router's location, as it was entered using the **snmp-server location** command. For example:

```
csh% getmany -v2c 10.10.11.12 public sysLocation
```

ENTITY-MIB

The following objects in the ENTITY-MIB provide basic information about the router's hardware:

- **entPhysicalDescr**—Provides a description of each hardware component in the router. For example, the following is a typical description for the Cisco uBR7246VXR chassis:

```
csh% getnext -v2c 10.10.11.12 public entPhysicalDescr
```

```
entPhysicalDescr.1 = uBR7246VXR chassis, Hw Serial#: 65100, Hw Revision: A
```

- **entPhysicalHardwareRev**—Provides the hardware revision of each component, if present and supported for that particular component. For example:

```
csh% getnext -v2c 10.10.11.12 public entPhysicalHardwareRev
```

```
entPhysicalHardwareRev.1 = 1.1
```

- **entPhysicalSerialNum**—Provides the serial number for each component, if present and supported for that particular component. For example:

```
csh% getnext -v2c 10.10.11.12 public entPhysicalSerialNum
```

```
entPhysicalSerialNum.1 = TBC06481339
```

- **entPhysicalModelName**—Provides the model name for each component, if present and supported for that particular component. For example:

```
csh% getnext -v2c 10.10.11.12 public entPhysicalModelName
```

```
entPhysicalModelName.1 = uBR7246VXR
```



Tip

Also see the next section for more information about the ENTITY-MIB and how to use it.

Managing Physical Components

The Cisco CMTS router supports a number of MIBs for the management of the router's physical components. These MIBs provide the following functions:

- Organizes the physical entities in the chassis into a containment tree that describes the relationship of each entity to all other entities

- Monitors and configures the status of field-replaceable units (FRUs)
- Maps physical ports to their respective interfaces
- Provides asset information for asset tagging
- Provides firmware and software information for chassis components

See the following sections for a description of each MIB, as well as instructions on how to use the MIBs to track the components in the router:

- [ENTITY-MIB, page A-5](#)
- [Cisco-Specific MIBS, page A-6](#)
- [Performing Inventory Management, page A-6](#)



Tip

To retrieve the chassis serial number for the router, retrieve the `chassisId` object from the `OLD-CISCO-CHASSIS-MIB`. This object defaults to the empty string, so you must use the `snmp-server chassis-id` command to set the value of this object before you can retrieve it.

ENTITY-MIB

The Cisco CMTS router uses the ENTITY-MIB, which is defined as the standard [RFC 2737](#), to manage its physical components, which are known as entities. An entity could be a card, a port on a card, a major subsystem on a card, a slot in the chassis, a field-replaceable unit (FRU), or any other equipment that is installed in the router.

The ENTITY-MIB defines a set of objects that uniquely identify each entity in the router, using a hierarchical containment tree that shows how each entity relates to each other. Other MIBs can then use the objects defined by the ENTITY-MIB to provide additional information about each entity.

The following are the most important objects in the ENTITY-MIB for the management of physical entities on the router:

- `entPhysicalTable`—Describes each physical component (entity) in the router. The table contains a row entry for the top-most entity (the chassis) and for each entity in the chassis. Each entry provides the name and description of the entry, its type and vendor, and a description of how the entity fits into the containment tree.
- `entPhysicalIndex`—Uniquely identifies each entry. This value is guaranteed to be unique across all equipment in this chassis and across all MIBs, allowing you to correlate the data from several MIBs for any particular entity.
- `entAliasMappingTable`—Maps each physical port's `entPhysicalIndex` value to the corresponding `ifIndex` value in the `ifTable` in the IF-MIB. This provides a quick way of identifying a particular port with a particular interface.

In Cisco IOS Release 12.2(15)BC2 and later releases, the `entAliasMappingTable` also shows the mapping of physical upstream connectors to logical upstream interface when virtual interfaces are configured on the Cisco uBR-MC5X20S and Cisco uBR-MC5X20U cable interface line cards. This support also changed the parent-child relationships of cable interface line cards and their ports. For more information, see the [“Changes to Support Virtual Interfaces” section on page 3-98](#).

- `entPhysicalContainsTable`—For each physical entity, lists the `entPhysicalIndex` value for any child objects of the entity. This provides an easy way of creating the container tree for the router, which shows the relationship between physical entities in the chassis.

Typically, the container tree is organized as follows:

- The chassis is the topmost level and contains the processor card and chassis slots.
- Chassis slots contain the individual line cards and I/O controller (if installed).
- Line cards contain ports (interfaces).
- Cable interface line cards contain downstream ports (known as cable line card, CLC). In Cisco IOS Release 12.2(15)BC1 and earlier releases, each downstream port then contains the upstream ports that are associated with it. In Cisco IOS Release 12.2(15)BC2 and later releases, the downstream ports and upstream are all children of the cable interface line card.

Cisco-Specific MIBS

In addition to the ENTITY-MIB, the Cisco CMTS router uses the following MIBs to provide additional information about the physical components that are installed in the router:

- CISCO-ENTITY-ASSET-MIB—Contains asset tracking information (ID PROM contents) for the physical entities listed in the entPhysicalTable of the ENTITY-MIB. The MIB provides device-specific information for physical entities, including orderable part number, serial number, and manufacturing assembly number, as well as hardware, software, and firmware information.
- CISCO-ENTITY-FRU-CONTROL-MIB—Contains objects used to monitor and configure the administrative and operational status of field-replaceable units (FRUs), such as power supplies and line cards, that are listed in the entPhysicalTable of the ENTITY-MIB.



Note Currently, the CISCO-ENTITY-FRU-CONTROL-MIB supports only line cards.

- CISCO-ENTITY-VENDORTYPE-OID-MIB—Contains the object identifiers (OIDs) for all physical entities in the router.
- CISCO-ENVMON-MIB—Contains information about the status of environmental sensors (for voltage, temperature, fans, and power supplies). For example, this MIB reports the chassis core and inlet temperatures.

Performing Inventory Management

The ENTITY-MIB provides all of the information needed to collect an inventory of the physical components in the router. The following procedure illustrates how to collect the inventory, using a Cisco uBR7246VXR router. In this example, the router contains the following cards:

- I/O Slot: Dual Fast Ethernet I/O Controller (FastEthernet0/0 and FastEthernet0/1)
- Slot 1: Fast Ethernet Line Card (FastEthernet1/0 and FastEthernet1/1)
- Slot 4: Cisco uBR-MC28U
- Slot 6: Cisco uBR-MC16E

To collect and organize the information in the ENTITY-MIB, use the following procedure.

-
- Step 1** Collect the list of physical entities by displaying all of the entPhysicalDescr objects. For example:

```

entPhysicalDescr.1 = uBR7246VXR chassis, Hw Serial#: 65100, Hw Revision: A
entPhysicalDescr.2 = NPE 400 Card
entPhysicalDescr.3 = Chassis Slot
entPhysicalDescr.4 = I/O Dual Fast Ethernet Controller
entPhysicalDescr.5 = i82543 (Livengood)
entPhysicalDescr.6 = i82543 (Livengood)
entPhysicalDescr.7 = Chassis Slot
entPhysicalDescr.8 = Dual Port Fast Ethernet (RJ45)
entPhysicalDescr.9 = i82543 (Livengood)
entPhysicalDescr.10 = i82543 (Livengood)
entPhysicalDescr.11 = Chassis Slot
entPhysicalDescr.12 = Chassis Slot
entPhysicalDescr.13 = Chassis Slot
entPhysicalDescr.14 = MC28U_F_connector
entPhysicalDescr.15 = UBR7200 CLC
entPhysicalDescr.16 = UBR7200 CLC
entPhysicalDescr.17 = BCM3138 PHY
entPhysicalDescr.18 = BCM3138 PHY
entPhysicalDescr.19 = BCM3138 PHY
entPhysicalDescr.20 = BCM3138 PHY
entPhysicalDescr.21 = BCM3034 PHY
entPhysicalDescr.22 = BCM3138 PHY
entPhysicalDescr.23 = BCM3138 PHY
entPhysicalDescr.24 = BCM3138 PHY
entPhysicalDescr.25 = BCM3138 PHY
entPhysicalDescr.26 = BCM3034 PHY
entPhysicalDescr.27 = Chassis Slot
entPhysicalDescr.28 = Chassis Slot
entPhysicalDescr.29 = MC16E
entPhysicalDescr.30 = BCM3210 ASIC
entPhysicalDescr.31 = BCM3137 PHY
entPhysicalDescr.32 = BCM3137 PHY
entPhysicalDescr.33 = BCM3137 PHY
entPhysicalDescr.34 = BCM3137 PHY
entPhysicalDescr.35 = BCM3137 PHY
entPhysicalDescr.36 = BCM3137 PHY
entPhysicalDescr.37 = BCM3033 PHY

```

- Step 2** Obtain additional information about each entPhysicalDescr object by collecting the entPhysicalVendorType, entPhysicalName, and entPhysicalClass objects. Use the index value to match the objects with their corresponding entPhysicalDescr object. [Table A-1](#) shows typical descriptions for the objects used in this example.

Table A-1 Sample entPhysicalDescr Objects and Descriptions

| Index # | entPhysicalDescr ¹ | entPhysicalVendorType | entPhysicalName | entPhysicalClass |
|---------|--|-----------------------|--------------------|------------------|
| 1 | uBR7246VXR chassis, Hw Serial#: 012345, Hw Revision: A | cevChassisUbr7246Vxr | | chassis(3) |
| 2 | NPE 400 Card | cevCpu7200Npe400 | | module(9) |
| 3 | Chassis Slot | cevContainerSlot | | container(5) |
| 4 | I/O Dual Fast Ethernet Controller | cevC7xxxIo2FE | | module(9) |
| 5 | i82543 (Livengood) | cevPortFEIP | FastEthernet0/0 | port(10) |
| 6 | i82543 (Livengood) | cevPortFEIP | FastEthernet0/1 | port(10) |
| 7 | Chassis Slot | cevContainerSlot | | container(5) |
| 8 | Dual Port Fast Ethernet (RJ45) | cevPa2feTxI82543 | | module(9) |
| 9 | i82543 (Livengood) | cevPortFEIP | FastEthernet1/0 | port(10) |
| 10 | i82543 (Livengood) | cevPortFEIP | FastEthernet1/1 | port(10) |
| 11 | Chassis Slot | cevContainerSlot | | container(5) |
| 12 | Chassis Slot | cevContainerSlot | | container(5) |
| 13 | Chassis Slot | cevContainerSlot | | container(5) |
| 14 | MC28U_F_connector | cevModuleUbrType | | module(9) |
| 15 | UBR7200 CLC | cevPortRfMac | Cable4/0 | port(10) |
| 16 | UBR7200 CLC | cevPortRfMac | Cable4/1 | port(10) |
| 17 | BCM3138 PHY | cevPortRfUs | Cable4/0-upstream0 | port(10) |

Table A-1 Sample entPhysicalDescr Objects and Descriptions (continued)

| Index # | entPhysicalDescr ¹ | entPhysicalVendorType | entPhysicalName | entPhysicalClass |
|---------|-------------------------------|-----------------------|---------------------|------------------|
| 18 | BCM3138 PHY | cevPortRfUs | Cable4/0-upstream1 | port(10) |
| 19 | BCM3138 PHY | cevPortRfUs | Cable4/0-upstream2 | port(10) |
| 20 | BCM3138 PHY | cevPortRfUs | Cable4/0-upstream3 | port(10) |
| 21 | BCM3034 PHY | cevPortRfDs | Cable4/0-downstream | port(10) |
| 22 | BCM3138 PHY | cevPortRfUs | Cable4/1-upstream0 | port(10) |
| 23 | BCM3138 PHY | cevPortRfUs | Cable4/1-upstream1 | port(10) |
| 24 | BCM3138 PHY | cevPortRfUs | Cable4/1-upstream2 | port(10) |
| 25 | BCM3138 PHY | cevPortRfUs | Cable4/1-upstream3 | port(10) |
| 26 | BCM3034 PHY | cevPortRfDs | Cable4/1-downstream | port(10) |
| 27 | Chassis Slot | cevContainerSlot | | container(5) |
| 28 | Chassis Slot | cevContainerSlot | | container(5) |
| 29 | MC16E | cevUbrMc16e | | module(9) |
| 30 | BCM3210 ASIC | cevPortRfMac | Cable6/0 | port(10) |
| 31 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream0 | port(10) |
| 32 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream1 | port(10) |
| 33 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream2 | port(10) |
| 34 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream3 | port(10) |
| 35 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream4 | port(10) |
| 36 | BCM3137 PHY | cevPortRfUs | Cable6/0-upstream5 | port(10) |
| 37 | BCM3033 PHY | cevPortRfDs | Cable6/0-downstream | port(10) |

1. Interfaces are typically identified by the chipset that is being used for the interface's connectors. On cable interfaces, upstreams are further identified by the chipsets that are providing the PHY-layer connectivity.

Step 3 To create the containment tree for the router, collect the EntPhysicalContainedIn object for each entPhysicalDescr object. The value in EntPhysicalContainedIn is the index number for the parent (or "container") for the corresponding entPhysicalDescr device.

The following shows the entPhysicalContainedIn values for the objects being used in this example:

```

entPhysicalContainedIn.1 = 0
entPhysicalContainedIn.2 = 1
entPhysicalContainedIn.3 = 1
entPhysicalContainedIn.4 = 3
entPhysicalContainedIn.5 = 4
entPhysicalContainedIn.6 = 4
entPhysicalContainedIn.7 = 1
entPhysicalContainedIn.8 = 7
entPhysicalContainedIn.9 = 8
entPhysicalContainedIn.10 = 8
entPhysicalContainedIn.11 = 1
entPhysicalContainedIn.12 = 1
entPhysicalContainedIn.13 = 1
entPhysicalContainedIn.14 = 13
entPhysicalContainedIn.15 = 14
entPhysicalContainedIn.16 = 14
entPhysicalContainedIn.17 = 15
entPhysicalContainedIn.18 = 15
entPhysicalContainedIn.19 = 15

entPhysicalContainedIn.20 = 15
entPhysicalContainedIn.21 = 15
entPhysicalContainedIn.22 = 16
entPhysicalContainedIn.23 = 16
entPhysicalContainedIn.24 = 16
entPhysicalContainedIn.25 = 16
entPhysicalContainedIn.26 = 16
entPhysicalContainedIn.27 = 1
entPhysicalContainedIn.28 = 1
entPhysicalContainedIn.29 = 28
entPhysicalContainedIn.30 = 29
entPhysicalContainedIn.31 = 30
entPhysicalContainedIn.32 = 30
entPhysicalContainedIn.33 = 30
entPhysicalContainedIn.34 = 30
entPhysicalContainedIn.35 = 30
entPhysicalContainedIn.36 = 30
entPhysicalContainedIn.37 = 30

```

Table A-2 shows the parent container for the entPhysicalDescr objects being used in this example.

Table A-2 Relationship of EntPhysicalContainedIn to entPhysicalDescr

| # | entPhysicalDescr | Is Contained In... | entPhysicalContainedInValue and the Parent Container |
|----|---|--------------------|--|
| 1 | uBR7246VXR chassis, Hw Serial#: 65100, Hw Revision: A | | 0, Topmost level, with no parent container |
| 2 | NPE 400 Card | | 1, Chassis |
| 3 | Chassis Slot | | 1, Chassis |
| 4 | I/O Dual Fast Ethernet Controller | | 3, Chassis Slot |
| 5 | i82543 (Livengood) | | 4, I/O Dual Fast Ethernet |
| 6 | i82543 (Livengood) | | 4, I/O Dual Fast Ethernet |
| 7 | Chassis Slot | | 1, Chassis |
| 8 | Dual Port Fast Ethernet (RJ45) | | 7, Chassis Slot |
| 9 | i82543 (Livengood) | | 8, Dual Port Fast Ethernet |
| 10 | i82543 (Livengood) | | 8, Dual Port Fast Ethernet |
| 11 | Chassis Slot | | 1, Chassis |
| 12 | Chassis Slot | | 1, Chassis |
| 13 | Chassis Slot | | 1, Chassis |
| 14 | MC28U_F_connector | | 13, Chassis Slot |
| 15 | UBR7200 CLC | | 14, MC28U_F connector |
| 16 | UBR7200 CLC | | 14, MC28U_F connector |
| 17 | BCM3138 PHY | | 15, UBR7200 CLC |
| 18 | BCM3138 PHY | | 15, UBR7200 CLC |

Table A-2 Relationship of *EntPhysicalContainedIn* to *entPhysicalDescr* (continued)

| # | entPhysicalDescr | Is Contained In... | entPhysicalContainedInValue and the Parent Container |
|----|------------------|--------------------|--|
| 19 | BCM3138 PHY | | 15, UBR7200 CLC |
| 20 | BCM3138 PHY | | 15, UBR7200 CLC |
| 21 | BCM3034 PHY | | 15, UBR7200 CLC |
| 22 | BCM3138 PHY | | 16, UBR7200 CLC |
| 23 | BCM3138 PHY | | 16, UBR7200 CLC |
| 24 | BCM3138 PHY | | 16, UBR7200 CLC |
| 25 | BCM3138 PHY | | 16, UBR7200 CLC |
| 26 | BCM3034 PHY | | 16, UBR7200 CLC |
| 27 | Chassis Slot | | 1, Chassis |
| 28 | Chassis Slot | | 1, Chassis |
| 29 | MC16E | | 28, Chassis Slot |
| 30 | BCM3210 ASIC | | 29, MC16E |
| 31 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 32 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 33 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 34 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 35 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 36 | BCM3137 PHY | | 30, BCM3210 ASIC |
| 37 | BCM3033 PHY | | 30, BCM3210 ASIC |

- Step 4** (Optional) If a parent object contains multiple children that are the same type of object, such as a router that contains multiple line card slots (Chassis Slots), use the *entPhysicalParentRelPos* objects to organize the child objects into their proper order. The *entPhysicalParentRelPos* objects contain an integer that shows the sequential order of the child objects. This integer typically starts incrementing from 0, so that it matches the actual numbering of the physical objects (slot 0 has an *entPhysicalParentRelPos* value of 0, slot 1 has an *entPhysicalParentRelPos* value of 1, and so forth).



Note If *entPhysicalParentRelPos* contains -1, then the object does not have an identifiable relationship with the other objects.

[Table A-3](#) shows how the *entPhysicalDescr* objects that refer to chassis slots can be put into their physical order by using their *entPhysicalParentRelPos* values. For example, *entPhysicalDescr.13* has an *entPhysicalParentRelPos* value of 4, indicating that this slot is slot 4/0 in the router chassis.

Table A-3 Using *entPhysicalParentRelPos* to Order *entPhysicalDescr* Objects

| # | entPhysicalDescr | entPhysicalContainedIn | entPhysicalParentRelPos | Physical Slot # |
|----|--------------------|------------------------|-------------------------|-----------------|
| 1 | uBR7246VXR chassis | 0 = TopLevel | -1 | N/A |
| 2 | NPE 400 Card | 1 = Chassis | -1 | None |
| 3 | Chassis Slot | 1 = Chassis | 0 | Slot 0/0 |
| 7 | Chassis Slot | 1 = Chassis | 1 | Slot 1/0 |
| 11 | Chassis Slot | 1 = Chassis | 2 | Slot 2/0 |
| 12 | Chassis Slot | 1 = Chassis | 3 | Slot 3/0 |
| 13 | Chassis Slot | 1 = Chassis | 4 | Slot 4/0 |
| 27 | Chassis Slot | 1 = Chassis | 5 | Slot 5/0 |
| 28 | Chassis Slot | 1 = Chassis | 6 | Slot 6/0 |

Step 5 (Optional) To map a physical interface to its *ifIndex*, which is defined in IF-MIB and used in other MIBs to uniquely identify a logical interface, use the *entAliasMappingIdentifier* object. If virtual interfaces are also configured on a cable interface line card, this table shows the mapping between the upstream's physical connector and its logical interface.

For example, the following shows the *entAliasMappingIdentifier* values for the router used in this example. In this example, *entPhysicalDescr.5* (which [Table A-1](#) identifies as the FastEthernet0/0 interface) maps to an *ifIndex* value of 1.

```
entAliasMappingIdentifier.5.0 = ifIndex.1
entAliasMappingIdentifier.6.0 = ifIndex.2
entAliasMappingIdentifier.9.0 = ifIndex.3
entAliasMappingIdentifier.10.0 = ifIndex.4
entAliasMappingIdentifier.15.0 = ifIndex.5
entAliasMappingIdentifier.16.0 = ifIndex.6
entAliasMappingIdentifier.17.0 = ifIndex.8
entAliasMappingIdentifier.18.0 = ifIndex.9
entAliasMappingIdentifier.19.0 = ifIndex.10
entAliasMappingIdentifier.20.0 = ifIndex.11
entAliasMappingIdentifier.21.0 = ifIndex.12
entAliasMappingIdentifier.22.0 = ifIndex.13
entAliasMappingIdentifier.23.0 = ifIndex.14
entAliasMappingIdentifier.24.0 = ifIndex.15
entAliasMappingIdentifier.25.0 = ifIndex.16
entAliasMappingIdentifier.26.0 = ifIndex.17
entAliasMappingIdentifier.30.0 = ifIndex.7
entAliasMappingIdentifier.31.0 = ifIndex.18
entAliasMappingIdentifier.32.0 = ifIndex.19
entAliasMappingIdentifier.33.0 = ifIndex.20
entAliasMappingIdentifier.34.0 = ifIndex.21
entAliasMappingIdentifier.35.0 = ifIndex.22
entAliasMappingIdentifier.36.0 = ifIndex.23
entAliasMappingIdentifier.37.0 = ifIndex.24
```

Generating SNMP Traps

This section describes how to configure the Cisco CMTS router to generate SNMP traps when certain events or conditions occur on the router. To use SNMP commands to configure the router to generate SNMP traps, you must define at least one target host to receive the traps, using the following procedure:



Tip

You can also use the command-line interface (CLI) to enable and configure the generation of traps on the router. For information on using the CLI, see the [“Enabling Notifications” section on page 4-2](#).

- Step 1** Create an entry in the `snmpTargetAddrTable`, which is defined in `SNMP-TARGET-MIB`, for each host that is to receive traps. Each entry contains the following objects:
- `snmpTargetAddrName`—Unique string, up to 32 characters long, that identifies this host.
 - `snmpTargetAddrTDomain`—The TCP/IP transport service to be used when delivering traps to this host, typically `snmpUDPDomain`.
 - `snmpTargetAddrTAddress`—The transport address for the host, typically a six-octet value that is composed of the four-byte IP address of the host followed by the two-byte UDP port number to which the traps should be sent.
 - `snmpTargetAddrTimeout`—Maximum period of time, in hundredths of a second, that the Cisco CMTS router waits for a response from the host (if any). The default is 1500 (15 seconds).
 - `snmpTargetAddrRetryCount`—Default number of times that the Cisco CMTS router resends a trap if a response is not received within the timeout period. The default value is three retries.
 - `snmpTargetAddrTagList`—List of tags (defined below) that should be associated with this particular target host. If a host’s tag value matches an `snmpNotifyTag` value, the host receives the types of notifications that are defined by the corresponding `snmpNotifyType`.
 - `snmpTargetAddrParams`—Arbitrary string, up to 32 characters long, that identifies an entry in the `snmpTargetParamsTable`, which defines the parameters to be used in generating traps.
 - `snmpTargetAddrStorageType`—Type of storage to be used for this row entry: `volatile(2)`, `nonVolatile(3)`, `permanent(4)`, or `readOnly(5)`. The default is `nonVolatile(4)`.
 - `snmpTargetAddrRowStatus`—Must be set to `createAndGo(4)` or `createAndWait(5)` to create this row entry. This object must be set only after all of the other entries in the row have been set.
- Step 2** Create an entry in the `snmpTargetParamsTable`, which is defined in `SNMP-TARGET-MIB`, to define the SNMP parameters that the router should use when generating SNMP notifications. Each entry contains the following objects:
- `snmpTargetParamsName`—Unique string, up to 32 characters long, that defines this particular entry. This string is also used in the `snmpTargetAddrParams` to define the parameters to be used when sending traps to any particular host.
 - `snmpTargetParamsMPModel`—Version of SNMP to be used in sending this trap: `SNMPv1(0)`, `SNMPv2c(1)`, and `SNMPv3(3)`.
 - `snmpTargetParamsSecurityModel`—Version of SNMP security to be used in sending traps: `SNMPv1(0)`, `SNMPv2c(1)`, and `SNMPv3(3)`.
 - `snmpTargetParamsSecurityName`—String, up to 32 characters long, to be used in identifying the Cisco CMTS router when sending traps.
 - `snmpTargetParamsSecurityLevel`—Type of security to be used when sending traps: `noAuthNoPriv(1)`, `authNoPriv(2)`, and `authPriv(3)`.

- `snmpTargetParamsStorageType`—Type of storage to be used for this row entry: `volatile(2)`, `nonVolatile(3)`, `permanent(4)`, or `readOnly(5)`. The default is `nonVolatile(4)`.
- `snmpTargetParamsRowStatus`—Must be set to `createAndGo(4)` or `createAndWait(5)` to create this row entry. This object must be set only after all of the other entries in the row have been set.

Step 3 Create an entry in the `snmpNotifyTable`, which is defined in the `SNMP-NOTIFICATION-MIB`. Each row in this table contains the following objects, which define a set of host targets that are to receive traps:

- `snmpNotifyName`—Unique string, up to 32 characters, that identifies this particular row entry.
- `snmpNotifyTag`—Arbitrary string, up to 255 characters, that identifies the set of hosts to receive traps. This tag value is matched against the `snmpTargetAddrTagList` object to determine which hosts should receive which traps.
- `snmpNotifyType`—Defines the type of trap to be set: `trap(1)` or `inform(2)`. The default is `trap(1)`.
- `snmpNotifyStorageType`—Type of storage to be used for this row entry: `volatile(2)`, `nonVolatile(3)`, `permanent(4)`, or `readOnly(5)`. The default is `nonVolatile(4)`.
- `snmpNotifyRowStatus`—Must be set to `createAndGo(4)` or `createAndWait(5)` to create this row entry. This object must be set only after all of the other entries in the row have been set.

Step 4 Optionally create rows in the `snmpNotifyFilterProfileTable` and `snmpNotifyFilterTable`, which are defined in the `SNMP-NOTIFICATION-MIB`. These tables create notification filters that limit the types of notifications that the router sends to particular hosts.

Step 5 Optionally enable traps and notifications to be sent. Most other MIBs include their own objects of `NOTIFICATION-TYPE` that enable or disable feature-specific traps. These notification objects also define the varbinds that are sent with each trap, which contain the specific information about the event that occurred.

For example, the [CISCO-CABLE-SPECTRUM-MIB](#) includes an `ccsHoppingNotification` object that enables or disables traps that are sent when a frequency hop occurs, or when the channel width or modulation profile changes. These traps provide the conditions of the upstream at the time of the hop, the old and new center frequencies, the old and new channel widths, and the old and new modulation profiles, as appropriate.

A number of notifications and traps can also be enabled using CLI commands. [Table A-4](#) lists some of the most common traps, how they can be enabled through the CLI, and the situations that generate these traps.

Table A-4 Common Notifications and Traps

| Type of Trap | Configuration Command to Enable | Description |
|------------------------------|--|--|
| Configuration Changes | <code>snmp-server enable traps entity</code> | <p>When ENTITY traps are enabled, the router generates an entConfigChange trap when the information in any of the following tables in the ENTITY-MIB changes:</p> <ul style="list-style-type: none"> entPhysicalTable entAliasMappingTable entPhysicalContainsTable <p>Note The SNMP manager should also regularly poll the entLastChangeTime object to detect whether traps were missed due to throttling or transmission loss.</p> |
| Environmental Changes | <code>snmp-server enable traps envmon</code> | <p>When ENVMON traps are enabled, the router generates the following traps (defined in CISCO-ENVMON-MIB) to notify you of potential environmental problems:</p> <ul style="list-style-type: none"> ciscoEnvMonShutdownNotification—Sent when the router is about to shut down. ciscoEnvMonTemperatureNotification—Sent when a temperature is outside its normal range. ciscoEnvMonFanNotification—Sent when a fan fails. ciscoEnvMonRedundantSupplyNotification—Sent when a redundant Power Entry Module fails. |
| FRU Status Changes | <code>snmp-server enable traps fru-ctrl</code> | <p>When FRU traps are enabled, the router generates the following traps (defined in CISCO-ENTITY-FRU-CONTROL-MIB) to notify the host of any changes in field-replaceable units (FRUs):</p> <ul style="list-style-type: none"> cefcModuleStatusChange—The operational status (cefcModuleOperStatus) of a FRU changed. cefcFRUInserted—A FRU was inserted in the chassis. cefcFRURemoved—A FRU was removed from the chassis. |
| Alarm is Asserted or Cleared | <code>snmp-server enable traps alarms</code> | <p>When ALARM traps are enabled, the router generates a trap whenever an alarm is asserted or cleared for physical entities that are defined in the entPhysicalTable in the ENTITY-MIB.</p> |
| SYSLOG Message is Generated | <code>snmp-server enable traps syslog</code> | <p>By default, the Cisco CMTS router logs a SYSLOG message each time an alarm is asserted or cleared. To also generate a separate trap when any SYSLOG message is logged, set the clogNotificationsEnabled object to true(1).</p> <p>Set the clogMaxSeverity object in CISCO-SYSLOG-MIB to the maximum severity level for the SYSLOG messages that are to be stored in the CISCO-SYSLOG-MIB and for which notifications should be generated. The default is 5 (warning), which indicates that SYSLOG messages of severity levels 1 through 5 are processed by the MIB.</p> |

Monitoring SYSLOG Messages

The CISCO-SYSLOG-MIB defines a number of objects that store the SYSLOG messages generated by Cisco CMTS router during its normal operations. You can regularly poll this MIB to obtain the list of SYSLOG messages that have been generated.

Message Table Objects

When enabled, SYSLOG messages are stored as an entry in the `clogHistoryTable`. Each `clogHistoryEntry` contains the following objects for each message that is stored:

- `clogHistIndex`—Index number that uniquely identifies each SYSLOG message that is stored in the table. This index is a 32-bit value that continually increases until it reaches its maximum value, at which point it wraps around back to 1.
- `clogHistFacility`—Facility identifier, up to 20 characters, of the SYSLOG message.
- `clogHistSeverity`—Severity level of the SYSLOG message, as defined by the `SyslogSeverity` textual convention, which ranges from 1 (emergency) to 8 (debug).



Note The severity numbers used in the `SyslogSeverity` and `clogHistSeverity` objects are one more than the numbers used in the actual SYSLOG messages. For example, an error SYSLOG message has a severity of 3, but `SyslogSeverity` uses 4 for error messages.

- `clogHistMsgName`—Mnemonic that identifies this SYSLOG message, up to 30 characters. If the mnemonic is longer than 30 characters, it is truncated to 29 characters and an asterisk (*) is appended to the end of the message to indicate that it has been truncated.
- `clogHistMsgText`—Actual text of the SYSLOG message, up to 255 characters, as it would appear in the console and SYSLOG logs. If a message is longer than 255 characters, it is truncated to 254 characters and an asterisk (*) is appended to the end of the message to indicate it has been truncated.
- `clogHistTimestamp`—Time stamp, in terms of `sysUpTime`, for when the SYSLOG message was generated.

Control Objects

The following objects in the CISCO-SYSLOG-MIB control the number and type of messages that are stored in the `clogHistoryTable`:

- `clogMaxSeverity`—Maximum severity level for the SYSLOG messages that are processed by this MIB. The default is 5 (warning), which indicates that SYSLOG messages of severity levels 1 through 5 are processed by the MIB.
- `clogMsgIgnores`—Number of SYSLOG messages that were ignored because it had a severity level greater than that specified by the `clogMaxSeverity`.
- `clogMsgDrops`—Number of SYSLOG messages that were dropped and not stored in the `clogHistoryTable` because of a lack of resources.
- `clogHistTableMaxLength`—Maximum number of SYSLOG messages that can be stored in the `clogHistoryTable`. When the table is full, the oldest message in the table is deleted to make room when a new SYSLOG message is generated. The valid range is 0 to 500, with a default of 1.

- `clogHistMsgsFlushed`—Number of entries that have been removed from the `clogHistoryTable` to make room for new entries. If this object is continually increasing, it indicates that you either need to increase the size of the table (`clogHistTableMaxLength`) or need to poll the table more frequently.

SYSLOG Notifications

You can configure the Cisco CMTS router so that it generates an SNMP notification when a SYSLOG message is generated. The notification sends an `clogMessageGenerated` object, which contains the following objects that identify the SYSLOG message:

- `clogHistFacility`
- `clogHistSeverity`
- `clogHistMsgName`
- `clogHistMsgText`
- `clogHistTimestamp`

To enable SYSLOG notifications using CLI commands, give the following command in global configuration mode:

```
snmp-server enable traps syslog
```

To enable these notifications using SNMP commands, set the `clogNotificationsEnabled` object to `true(1)`. The `clogNotificationsSent` object then contains the number of `clogMessageGenerated` notifications that have been sent.

Example

The following example shows typical output from the CISCO-SYSLOG-MIB when using the SNMP utilities that are standard on many UNIX-based systems. This router uses the default configuration, where only one SYSLOG message is stored in the `clogHistoryTable`. The table currently contains an entry with the index of 25, and `clogHistMsgsFlushed` shows that the 24 previous messages have already been flushed from the table.

```
csh% getmany -v2c 10.10.11.12 public ciscoSyslogMIB

clogNotificationsSent.0 = 0
clogNotificationsEnabled.0 = false(2)
clogMaxSeverity.0 = warning(5)
clogMsgIgnores.0 = 199
clogMsgDrops.0 = 0
clogHistTableMaxLength.0 = 1
clogHistMsgsFlushed.0 = 24
clogHistFacility.25 = UBR7200
clogHistSeverity.25 = error(4)
clogHistMsgName.25 = AUTH_REJECT_PERMANENT_AUTHORI*
clogHistMsgText.25 = <132>CMTS[DOCSIS]:<66030108> Auth Reject - Permanent
Authorization Failure . CM Mac Addr <000C.AB01.CD89>
clogHistTimestamp.25 = 4452551

csh%
```

Displaying Information About Cable Modems

This section describes how to obtain information about one or all cable modems:

- “Displaying Current Status for Cable Modems” section on page A-17
- “Displaying Information About Burst and Station Maintenance Intervals” section on page A-18
- “Logging and Displaying Deleted Service Flows” section on page A-19

Displaying Current Status for Cable Modems

To display the current status for one or more cable modems, display the docsIfCmtsCmStatusValue object from the docsIfCmtsCmStatusTable, which is in the DOCS-IF-MIB. This object contains the following states for each cable modem:

- other(1)—Any state other than those listed below, such as offline.
- ranging(2)—The cable modem has sent an initial ranging request to the Cisco CMTS, but has not yet completed the ranging process.
- rangingAborted(3)—The Cisco CMTS has sent a Ranging Abortion message to the cable modem, requiring it to restart the ranging process.
- rangingComplete(4)—The Cisco CMTS has sent a Ranging Complete message to the cable modem, allowing it to continue on to the registration process.
- ipComplete(5)—The cable modem has sent a DHCP broadcast, and the Cisco CMTS has forwarded the DHCP reply, with an assigned IP address, to the cable modem.
- registrationComplete(6)—The Cisco CMTS has sent a Registration Response message to the cable modem, indicating that the cable modem has completed the registration process and can now come online and forward traffic from its CPE devices.
- accessDenied(7) —The Cisco CMTS has sent a Registration Aborted message to the cable modem, indicating that the provisioning system is not allowing the cable modem and its CPE devices to come online.

To display the current status of one or more cable modems, use the following procedure:

- Step 1** Poll the docsIfCmtsCmStatusMacAddress object from the docsIfCmtsCmStatusTable to obtain the MAC addresses for the known cable modems:

```
csh% getmany -v2c 10.10.17.91 public docsIfCmtsCmStatusMacAddress

docsIfCmtsCmStatusMacAddress.671745 = 00 0a ff 01 44 5e
docsIfCmtsCmStatusMacAddress.671746 = 00 0b fe 01 18 5e
docsIfCmtsCmStatusMacAddress.671747 = 00 0c fd 21 bb 54
docsIfCmtsCmStatusMacAddress.671748 = 00 0d fc 89 5f a9
docsIfCmtsCmStatusMacAddress.671749 = 00 0e fb 89 6b fd
docsIfCmtsCmStatusMacAddress.671750 = 00 0f fa 89 5c 6d
docsIfCmtsCmStatusMacAddress.671751 = 00 00 f0 89 5d 35

csh%
```

- Step 2** Poll the docsIfCmtsCmStatusValue objects to obtain the current status of those cable modems:

```
csh% getmany -v2c 10.10.17.91 public docsIfCmtsCmStatusValue

docsIfCmtsCmStatusValue.671745 = registrationComplete(6)
```

```
docsIfCmtsCmStatusValue.671746 = registrationComplete(6)
docsIfCmtsCmStatusValue.671747 = registrationComplete(6)
docsIfCmtsCmStatusValue.671748 = registrationComplete(6)
docsIfCmtsCmStatusValue.671749 = accessDenied(7)
docsIfCmtsCmStatusValue.671750 = registrationComplete(6)
docsIfCmtsCmStatusValue.671751 = registrationComplete(6)
```

```
csch%
```

- Step 3** Use the index value of the docsIfCmtsCmStatusValue and docsIfCmtsCmStatusMacAddress objects to find the current status of any particular cable modem. For example, the output displayed by this example shows that the cable modem with the MAC address of 00.0E.FB.89.6B.FD, which has the index of 671749, is currently marked with a status of accessDenied(7).

```
docsIfCmtsCmStatusMacAddress.671749 = 00 0e fb 89 6b fd
...
docsIfCmtsCmStatusValue.671749 = accessDenied(7)
```

Displaying Information About Burst and Station Maintenance Intervals

To display information about the burst and station maintenance intervals being used on an upstream, poll the cdxUpInfoElemStatsIEType object from the CISCO-DOCS-EXT-MIB. This object is a sequence of six entries that are indexed by the upstream's ifDescr value:

- cdxUpInfoElemStatsIEType.*upstream-ifDescr.1*—Displays the current number of mini-slots being used for request bursts (reqIE), which are used for bandwidth requests.
- cdxUpInfoElemStatsIEType.*upstream-ifDescr.2*—Displays the current number of mini-slots being used for request/data bursts (reqOrDataIE), which are used for bandwidth or short data packet requests.
- cdxUpInfoElemStatsIEType.*upstream-ifDescr.3*—Displays the current number of mini-slots being used for initial maintenance bursts (initMtnIE), which are reserved for new cable modems that want to come online.
- cdxUpInfoElemStatsIEType.*upstream-ifDescr.4*—Displays the current number of mini-slots being used for station maintenance bursts (stnMtnIE), which are used for keepalive and network maintenance messages.
- cdxUpInfoElemStatsIEType.*upstream-ifDescr.5*—Displays the current number of mini-slots being used for short data grant bursts (shortGrantIE), which are used for short data grants.
- cdxUpInfoElemStatsIEType.*upstream-ifDescr.6*—Displays the current number of mini-slots being used for long data grant bursts (longGrantIE), which are used for large data requests.

To obtain this information, use the following procedure:

- Step 1** Obtain the ifIndex for the desired upstream. This can be done by requesting a GET request for ifDescr. For example, the following shows sample output from a Cisco uBR7246VXR router that has one Fast Ethernet port adapter and one Cisco uBR-MC16C cable interface card installed:

```
csch% getmany -v2c 10.10.10.13 public ifDescr

ifDescr.1 = FastEthernet0/0
ifDescr.2 = FastEthernet0/1
ifDescr.3 = FastEthernet1/0
ifDescr.4 = FastEthernet1/1
ifDescr.5 = Cable4/0
```

```

ifDescr.8 = Cable4/0-upstream0
ifDescr.9 = Cable4/0-upstream1
ifDescr.10 = Cable4/0-upstream2
ifDescr.11 = Cable4/0-upstream3
ifDescr.12 = Cable4/0-upstream4
ifDescr.13 = Cable4/0-upstream5

```

```
csh%
```

Step 2 Use the ifIndex for the desired upstream as the index for each of the six cdxUpInfoElemStatsIEType objects. The following example uses ifDescr for upstream 0 on the Cisco uBR-MC16C card:

```
csh% getmany -v2c 10.10.10.13 public cdxUpInfoElemStatsIEType.8
```

```

cdxUpInfoElemStatsIEType.8.reqIE = 76826109
cdxUpInfoElemStatsIEType.8.reqOrDataIE = 0
cdxUpInfoElemStatsIEType.8.initMtnIE = 494562
cdxUpInfoElemStatsIEType.8.stnMtnIE = 47447
cdxUpInfoElemStatsIEType.8.shortGrantIE = 242
cdxUpInfoElemStatsIEType.8.longGrantIE = 29116

```

```
csh%
```

**Note**

The Cisco CMTS supports, but does not use, request/data bursts, so the output for the reqOrDataIE type is always 0.

Logging and Displaying Deleted Service Flows

The DOCSIS 2.0 specifications require that the CMTS maintains a log table of deleted DOCSIS 1.1 or DOCSIS 2.0 service flows. Entries stay in this table until they either age out, or until the table becomes full, at which time the oldest entries are deleted to make room for the newest ones.

Use the following procedure to enable logging of deleted service flows, and then to display the entries in the log table (docsQosServiceFlowLogTable in the [DOCS-QOS-MIB](#)).

Step 1 On the Cisco CMTS router console, enable logging of deleted service flows by giving the **cable sflog** command at the global configuration prompt. This command has the following syntax:

```
cable sflog max-entry number entry-duration time
```

You must specify the following parameters:

- **max-entry** *number*—Specifies the maximum number of entries in the service flow log. When the log becomes full, the oldest entries are deleted to make room for new entries. The valid range is 0 to 59999, with a default of 0 (which disables service flow logging).

**Note**

The **max-entry** value applies to the entire chassis on the Cisco uBR7100 series and Cisco uBR7200 series routers, but applies to individual cable line cards on the Cisco uBR10012 router.

- **entry-duration** *time*—Specifies how long, in seconds, entries can remain in the service flow log. The CMTS deletes entries in the log that are older than this value. The valid range is 1 to 86400 seconds, with a default value of 3600 seconds (1 hour).

For example, the following enables logging of deleted service flows with a table that has room for 20,000 entries, and that automatically deletes entries after they have been in the table for two hours:

```
Router(config)# cable sflow max-entry 20000 entry-duration uBR7200
```

Step 2 At regular intervals, the network management station should poll the docsQosServiceFlowLogTable to collect the information about the deleted service flows. The table entries are indexed by a unique 32-bit index, which wraps around to 0 when it reaches its maximum value.

The following example shows that the docsQosServiceFlowLogTable contains two entries for two deleted service flows:

```
csh% getmany -v2c 10.17.16.1 public docsQosServiceFlowLogTable
docsQosServiceFlowLogIfIndex.180001 = 10
docsQosServiceFlowLogIfIndex.180002 = 10
docsQosServiceFlowLogSFID.180001 = 3
docsQosServiceFlowLogSFID.180002 = 4
docsQosServiceFlowLogCmMac.180001 = 00 00 39 42 b2 56
docsQosServiceFlowLogCmMac.180002 = 00 00 39 42 b2 56
docsQosServiceFlowLogPkts.180001 = 0
docsQosServiceFlowLogPkts.180002 = 0
docsQosServiceFlowLogOctets.180001 = 0
docsQosServiceFlowLogOctets.180002 = 0
docsQosServiceFlowLogTimeDeleted.180001 = 58800
docsQosServiceFlowLogTimeDeleted.180002 = 58800
docsQosServiceFlowLogTimeCreated.180001 = 9400
docsQosServiceFlowLogTimeCreated.180002 = 9400
docsQosServiceFlowLogTimeActive.180001 = 474
docsQosServiceFlowLogTimeActive.180002 = 474
docsQosServiceFlowLogDirection.180001 = upstream(2)
docsQosServiceFlowLogDirection.180002 = downstream(1)
docsQosServiceFlowLogPrimary.180001 = true(1)
docsQosServiceFlowLogPrimary.180002 = true(1)
docsQosServiceFlowLogServiceClassName.180001 =
docsQosServiceFlowLogServiceClassName.180002 =
docsQosServiceFlowLogPolicedDropPkts.180001 = 0
docsQosServiceFlowLogPolicedDropPkts.180002 = 0
docsQosServiceFlowLogPolicedDelayPkts.180001 = 8
docsQosServiceFlowLogPolicedDelayPkts.180002 = 0
docsQosServiceFlowLogControl.180001 = active(1)
docsQosServiceFlowLogControl.180002 = active(1)

csh%
```

Monitoring Spectrum Management

When you are using Cisco IOS Release 12.2(8)BC2 and later releases, you can use SNMP to access the CISCO-CABLE-SPECTRUM-MIB to monitor the spectrum management activity on cable interface cards, such as the Cisco uBR-MC16S, that include a hardware-based spectrum analyzer. You can use the MIB to perform the following tasks:

- [Enabling Spectrum Management, page A-21](#)
- [Displaying the Results of a Spectrum Request, page A-23](#)

- [Monitoring CNR Measurements for Individual Cable Modems, page A-24](#)
- [Displaying Information About Frequency Hops, page A-26](#)

Enabling Spectrum Management

To enable spectrum management for an upstream on a Cisco uBR-MC16C, Cisco uBR-MC16U/X, Cisco uBR-MC28U/X, or Cisco uBR-MC5X20S/U cable interface card, create an entry in the `ccsSpectrumRequestTable` for the upstream. [Table A-5](#) lists the objects that can be configured for each entry in the `ccsSpectrumRequestTable` table.

Table A-5 *ccsSpectrumRequestTable* Attributes

| Attribute | Type | Description |
|--|--------------------------|--|
| <code>ccsSpectrumRequestIndex</code> | Integer32 | Arbitrary index to uniquely identify each table entry. |
| <code>ccsSpectrumRequestIfIndex</code> | InterfaceIndex OrZero | IfIndex identifying an upstream on a cable interface line card that supports hardware spectrum management. |
| <code>ccsSpectrumRequestMacAddr</code> | MacAddress | MAC address to request a signal-to-noise ratio (SNR) or carrier-to-noise ration (CNR) value for a particular cable modem, or 0000.0000.0000 to request background noise for the entire spectrum. |
| <code>ccsSpectrumRequestUpperFreq</code> | CCSFrequency | Upper frequency (in KHz) for the frequency range to be monitored (5000 to 42000 KHz, with a default of 42000 KHz). |
| <code>ccsSpectrumRequestLowFreq</code> | CCSFrequency | Lower frequency (in KHz) for the frequency range to be monitored (5000 to 42000 KHz, with a default of 5000 KHz). |
| <code>ccsSpectrumRequestResolution</code> | Integer32 | Requested resolution to determine how the frequency range should be sampled (12 to 37000 KHz, with a default of 60 KHz). |
| <code>ccsSpectrumRequestStartTime</code> | TimeStamp | Time when the spectrum measurement began. |
| <code>ccsSpectrumRequestStoppedTime</code> | TimeStamp | Time when the spectrum measurement finished. |
| <code>ccsSpectrumRequestOperation</code> | CCSRequestOp eration | Starts a new spectrum management request or aborts the current one. |
| <code>ccsSpectrumRequestOperState</code> | CCSRequestOp erState | Provides the operational state of the current spectrum management request. |
| <code>ccsSpectrumRequestStatus</code> | RowStatus | Controls the modification, creation, and deletion of table entries. |

To enable spectrum management on an upstream, use the following procedure:

- Step 1** Obtain the ifIndex for the desired upstream. This can be done by requesting a GET request for ifDescr. For example, the following shows sample output from a Cisco uBR7246VXR router that has one Fast Ethernet port adapter and one Cisco uBR-MC16S cable interface card installed in slot 6:

```
csh% getmany -v2c 10.10.10.13 public ifDescr

ifDescr.1 = FastEthernet0/0
ifDescr.2 = FastEthernet0/1
ifDescr.3 = FastEthernet1/0
ifDescr.4 = FastEthernet1/1
ifDescr.5 = Cable6/0
ifDescr.6 = Cable6/0-upstream0
ifDescr.7 = Cable6/0-upstream1
ifDescr.8 = Cable6/0-upstream2
ifDescr.9 = Cable6/0-upstream3
ifDescr.10 = Cable6/0-upstream4
ifDescr.11 = Cable6/0-upstream5

csh%
```

- Step 2** Create a row in the ccsSpectrumRequestTable for the desired upstream. At a minimum, you must create the row in the wait condition and then configure the upstream ifIndex before activating the row.

The following example shows a row being created for upstream 2 (ifIndex 8). The arbitrary row index of 8 has been chosen for this row—ensure that the row index you choose is not currently in use.

```
csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestStatus.8 -i 5
ccsSpectrumRequestStatus.8 = wait(5)

csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestIfIndex.8 -i 8
ccsSpectrumRequestIfIndex.8 = 8

csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestStatus.8 -i 1
ccsSpectrumRequestStatus.8 = active(1)

csh%
```



Note If you attempt to use an IfIndex that does not specify an upstream on a cable interface line card that supports hardware spectrum management, the SET request fails with an invalid value error.

- Step 3** Display the current configuration of the new row entry, to verify that the default parameters are acceptable:

```
csh% getmany -v2c 10.10.10.13 public ccsSpectrumRequestTable

ccsSpectrumRequestIfIndex.8 = 8
ccsSpectrumRequestMacAddr.8 = 00 00 00 00 00 00
ccsSpectrumRequestLowFreq.8 = 5000
ccsSpectrumRequestUpperFreq.8 = 42000
ccsSpectrumRequestResolution.8 = 60
ccsSpectrumRequestOperation.8 = none(0)
ccsSpectrumRequestOperState.8 = idle(0)
ccsSpectrumRequestStartTime.8 = 0
ccsSpectrumRequestStoppedTime.8 = 0
ccsSpectrumRequestStatus.8 = active(1)

csh%
```

- Step 4** By default, `ccsSpectrumRequestMacAddr` is set to all zeros, which requests background noise for the entire upstream. To monitor the CNR for a particular cable modem, set `ccsSpectrumRequestMacAddr` to its MAC address:

```
csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestMacAddr.8 -o '00 01 64 ff eb 95'
ccsSpectrumRequestMacAddr.3 = 00 01 64 ff eb 95
```

```
csh% getmany -v2c 10.10.10.13 public ccsSpectrumRequestTable
```

```
ccsSpectrumRequestIfIndex.8 = 8
ccsSpectrumRequestMacAddr.8 = 00 01 64 ff eb 95
ccsSpectrumRequestLowFreq.8 = 5000
ccsSpectrumRequestUpperFreq.8 = 42000
ccsSpectrumRequestResolution.8 = 60
ccsSpectrumRequestOperation.8 = none(0)
ccsSpectrumRequestOperState.8 = idle(0)
ccsSpectrumRequestStartTime.8 = 0
ccsSpectrumRequestStoppedTime.8 = 0
ccsSpectrumRequestStatus.8 = active(1)
```

```
csh%
```

- Step 5** If any other parameters need to be changed from their default values, change them to the desired values. For example, the following shows the frequency range being changed from the default range (5 MHz to 42 MHz) to 20 MHz to 28 MHz:

```
csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestLowFreq.8 -i 20000
ccsSpectrumRequestLowFreq.8 = 20000
```

```
csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestUpperFreq.8 -i 28000
ccsSpectrumRequestUpperFreq.8 = 28000
```

```
csh%
```

- Step 6** When all parameters are correct, set the `ccsSpectrumRequestOperation` object to `start(1)` to begin spectrum monitoring of the upstream:

```
csh% setany -v2c 10.10.10.13 private ccsSpectrumRequestOperation.8 -i 1
ccsSpectrumRequestOperation.8 = start(1)
```

```
csh%
```

Displaying the Results of a Spectrum Request

To monitor the results of a spectrum request, display the objects in the corresponding row in the `ccsSpectrumDataTable` (which displays the same information as that shown by the **show controllers cable upstream spectrum** command). [Table A-6](#) lists the objects that are stored in this table:

Table A-6 *ccsSpectrumDataTable Attributes*

| Attribute | Type | Description |
|-----------------------------------|-----------------------|---|
| <code>ccsSpectrumDataFreq</code> | CCSMeasured Frequency | Frequency in KHz for which this power measurement was made. |
| <code>ccsSpectrumDataPower</code> | INTEGER | Measured received power for the given frequency (-50 to 50 dBmV). |

To display the results of a spectrum request, use the following procedure:

Step 1 Create and activate a spectrum request by adding a row to the `ccsSpectrumRequestTable`, as shown in the “[Enabling Spectrum Management](#)” section on page A-21.

Step 2 Display the entries in the `ccsSpectrumDataTable`:

```
csh% getmany -v2c 10.10.10.13 public ccsSpectrumDataTable

ccsSpectrumDataFreq.8.20001 = 20001
ccsSpectrumDataFreq.8.20121 = 20121
ccsSpectrumDataFreq.8.20241 = 20241
ccsSpectrumDataFreq.8.20361 = 20361

...

ccsSpectrumDataFreq.8.27561 = 27561
ccsSpectrumDataFreq.8.27681 = 27681
ccsSpectrumDataFreq.8.27801 = 27801
ccsSpectrumDataFreq.8.27921 = 27921

ccsSpectrumDataPower.8.20001 = -43
ccsSpectrumDataPower.8.20121 = -50
ccsSpectrumDataPower.8.20241 = -47
ccsSpectrumDataPower.8.20361 = -46

...

ccsSpectrumDataPower.8.27561 = -47
ccsSpectrumDataPower.8.27681 = -44
ccsSpectrumDataPower.8.27801 = -46
ccsSpectrumDataPower.8.27921 = -42

csh%
```



Note

The entries in the `ccsSpectrumDataTable` are indexed by the row number for the spectrum request entry in the `ccsSpectrumRequestTable` and by the frequency (in KHz) at which the power measurement was made.

Step 3 Use the `ccsSpectrumDataFreq` values to determine the frequencies at which power measurements were made. Then use the frequency value, as well as the `ccsSpectrumRequestTable` row entry, to determine the specific power measurement for any particular frequency.

For example, the following line shows that the power measurement for the upstream specified in row 3 of the `ccsSpectrumRequestTable` is -46 dBmV at 27.801 MHz:

```
ccsSpectrumDataPower.8.27801 = -46
```

Monitoring CNR Measurements for Individual Cable Modems

To obtain the carrier-to-noise ratio (CNR) for an individual cable modem, create and activate an entry in the `ccsSNRRRequestTable`. [Table A-7](#) lists the objects that can be configured for each entry.

Table A-7 *ccsSNRRRequestTable Attributes*

| Attribute | Type | Description |
|---------------------------|---------------------|---|
| ccsSNRRRequestIndex | Integer32 | Arbitrary index to uniquely identify each table entry. |
| ccsSNRRRequestMacAddr | MacAddress | MAC address of the remote online cable modem being reported on. |
| ccsSNRRRequestSNR | Integer32 | SNR value, in dB, that has been measured. This value is 0 when the operation state is "running." |
| ccsSNRRRequestOperation | CCSRequestOperation | Sets the current operation: start, pending, running, or abort. |
| ccsSNRRRequestOperState | CCSRequestOperState | Reports on the current operation state: idle, pending, running, noError, aborted, notOnLine, invalidMac, timeOut, fftBusy, fftFailed, others. |
| ccsSNRRRequestStartTime | TimeStamp | Contains the time when the SNR measurement operation starts. |
| ccsSNRRRequestStoppedTime | TimeStamp | Contains the time when the SNR measurement stops. |
| ccsSNRRRequestStatus | RowStatus | Controls the modification, creation, and deletion of table entries. |

To obtain CNR information for a particular cable modem, use the following procedure:

- Step 1** Create a row in the `ccsSNRRRequestTable` for the desired cable modem. At a minimum, you must create the row in the wait condition and then configure the cable modem's MAC address before activating the row. For example:

```
csh% setany -v2c 10.10.10.13 private ccsSNRRRequestStatus.200 -i 5
ccsSNRRRequestStatus.200 = createAndWait(5)

csh% setany -v2c 10.10.10.13 private ccsSNRRRequestMacAddr.200 -o '00 03 e3 50 9b 3d'
ccsSNRRRequestMacAddr.200 = 00 03 e3 50 9b 3d

csh% setany -v2c 10.10.10.13 private ccsSNRRRequestStatus.200 -i 1
ccsSNRRRequestStatus.200 = active(1)

csh%
```

- Step 2** Display the current configuration of the new row entry, to verify that the default parameters are acceptable:

```
csh% getmany -v2c 10.10.10.13 public ccsSNRRRequestTable

ccsSNRRRequestMacAddr.200 = 00 03 e3 50 9b 3d
ccsSNRRRequestSNR.200 = 0
ccsSNRRRequestOperation.200 = none(0)
ccsSNRRRequestOperState.200 = idle(0)
ccsSNRRRequestStartTime.200 = 0
ccsSNRRRequestStoppedTime.200 = 0
ccsSNRRRequestStatus.200 = active(1)

csh%
```

- Step 3** When all parameters are correct, set the `ccsSpectrum ccsSNRRequestOperation` object to `start(1)` to begin monitoring of the cable modem:

```
csh% setany -v2c 10.10.10.13 private ccsSNRRequestOperation.200 -i 1
ccsSNRRequestOperation.200 = start(1)
```

```
csh%
```

- Step 4** Repeatedly poll the `ccsSNRRequestTable` until `ccsSNRRequestOperState` shows `noError`, at which point `ccsSNRRequestSNR` shows the cable modem's current CNR value:



Note The `ccsSNRRequestSNR` object continues to show 0 as long as `ccsSNRRequestOperState` shows the state as `running(2)`.

```
csh% getmany -v2c 10.10.10.13 public ccsSNRRequestTable
```

```
ccsSNRRequestMacAddr.200 = 00 03 e3 50 9b 3d
ccsSNRRequestSNR.200 = 0
ccsSNRRequestOperation.200 = start(1)
ccsSNRRequestOperState.200 = running(2)
ccsSNRRequestStartTime.200 = 0
ccsSNRRequestStoppedTime.200 = 0
ccsSNRRequestStatus.200 = active(1)
```

```
csh% getmany -v2c 10.10.10.13 public ccsSNRRequestTable
```

```
ccsSNRRequestMacAddr.200 = 00 03 e3 50 9b 3d
ccsSNRRequestSNR.200 = 25
ccsSNRRequestOperation.200 = start(1)
ccsSNRRequestOperState.200 = noError(3)
ccsSNRRequestStartTime.200 = 298853
ccsSNRRequestStoppedTime.200 = 298974
ccsSNRRequestStatus.200 = active(1)
```

```
csh%
```

Displaying Information About Frequency Hops

To obtain information about the most recent frequency hops on an upstream, display the objects in the `ccsUpSpecMgmtTable`. [Table A-8](#) lists the `ccsUpSpecMgmtEntry` attributes.

Table A-8 *ccsUpSpecMgmtEntry Attributes*

| Attribute | Type | Description |
|---------------------------------------|-----------|---|
| <code>ccsUpSpecMgmtHopPriority</code> | INTEGER | Specifies the priority of frequency, modulation profile, and channel width in determining corrective action for excessive noise on the upstream (default is frequency, modulation profile, and channel width) |
| <code>ccsUpSpecMgmtSnrThres1</code> | Integer32 | Specifies the upper SNR threshold for modulation profile 1 (5–35 dB, default of 25) |

Table A-8 *ccsUpSpecMgmtEntry Attributes (continued)*

| Attribute | Type | Description |
|--|--------------|---|
| ccsUpSpecMgmtSnrThres2 | Integer32 | Specifies the upper SNR threshold for modulation profile 2 (5–35 dB, default of 15, and must be lower than that specified for ccsUpSpecMgmtSnrThres1) |
| ccsUpSpecMgmtFecCorrectThres1 | Integer32 | Specifies the forward error correction (FEC) correctable error threshold for modulation profile 1 (1–20%) |
| ccsUpSpecMgmtFecCorrectThres2 | Integer32 | Deprecated and no longer used |
| ccsUpSpecMgmtFecUnCorrectThres1 | Integer32 | Specifies the FEC uncorrectable error threshold for modulation profile 1 (1–20%) |
| ccsUpSpecMgmtFecUnCorrectThres2 | Integer32 | Deprecated and no longer used |
| ccsUpSpecMgmtSnrPollPeriod | Integer32 | Deprecated and no longer used |
| ccsUpSpecMgmtHopCondition ¹ | INTEGER | Reports the condition that triggers a frequency hop (SNR value or percentage of modems going offline) |
| ccsUpSpecMgmtFromCenterFreq ¹ | CCSFrequency | Provides the center frequency (in KHz) before the latest frequency hop |
| ccsUpSpecMgmtToCenterFreq ¹ | CCSFrequency | Provides the current center frequency (in KHz) after the latest frequency hop |
| ccsUpSpecMgmtFromBandWidth ¹ | CCSFrequency | Provides the channel width (in KHz) before the latest frequency hop |
| ccsUpSpecMgmtToBandWidth ¹ | CCSFrequency | Provides the current channel width (in KHz) after the latest frequency hop |
| ccsUpSpecMgmtFromModProfile ¹ | Integer32 | Provides the modulation profile number before the latest frequency hop |
| ccsUpSpecMgmtToModProfile ¹ | Integer32 | Provides the current modulation profile number after the latest frequency hop |
| ccsUpSpecMgmtSNR | Integer32 | Provides the current SNR value (in dB) for the upstream |

1. These objects are also sent in the notification message that is sent when a frequency hop occurs on an upstream.

To collect the frequency hopping data for one or more upstreams, use the following procedure:

- Step 1** Obtain the ifIndex for the desired upstreams. This can be done by requesting a GET request for ifDescr. For example, the following shows sample output from a Cisco uBR7246VXR router with the first four upstreams on the cable interface card in slot 5/0:

```

csh% getmany -v2c 10.10.10.13 public ifDescr
...

ifDescr.24 = Cable5/0-upstream0
ifDescr.25 = Cable5/0-upstream1
ifDescr.26 = Cable5/0-upstream2
ifDescr.27 = Cable5/0-upstream3

```

```
...
```

```
csh%
```

- Step 2** Display the `ccsUpSpecMgmtTable`. Use the `ifDescr` values of the desired upstreams to find the values for those upstreams. The following example shows the relevant output for the four upstreams shown above.

```
csh% getmany -v2c 10.10.10.13 public ccsUpSpecMgmtTable
```

```
...
ccsUpSpecMgmtHopPriority.24 = frqModChannel(0)
ccsUpSpecMgmtHopPriority.25 = frqModChannel(0)
ccsUpSpecMgmtHopPriority.26 = frqModChannel(0)
ccsUpSpecMgmtHopPriority.27 = frqModChannel(0)
...
ccsUpSpecMgmtSnrThres1.24 = 25
ccsUpSpecMgmtSnrThres1.25 = 25
ccsUpSpecMgmtSnrThres1.26 = 25
ccsUpSpecMgmtSnrThres1.27 = 25
...
ccsUpSpecMgmtSnrThres2.24 = 15
ccsUpSpecMgmtSnrThres2.25 = 15
ccsUpSpecMgmtSnrThres2.26 = 15
ccsUpSpecMgmtSnrThres2.27 = 15
...
ccsUpSpecMgmtFecCorrectThres1.24 = 1
ccsUpSpecMgmtFecCorrectThres1.25 = 1
ccsUpSpecMgmtFecCorrectThres1.26 = 1
ccsUpSpecMgmtFecCorrectThres1.27 = 1
...
ccsUpSpecMgmtFecCorrectThres2.24 = 1
ccsUpSpecMgmtFecCorrectThres2.25 = 1
ccsUpSpecMgmtFecCorrectThres2.26 = 1
ccsUpSpecMgmtFecCorrectThres2.27 = 1
...
ccsUpSpecMgmtFecUnCorrectThres1.24 = 1
ccsUpSpecMgmtFecUnCorrectThres1.25 = 1
ccsUpSpecMgmtFecUnCorrectThres1.26 = 1
ccsUpSpecMgmtFecUnCorrectThres1.27 = 1
...
ccsUpSpecMgmtFecUnCorrectThres2.24 = 1
ccsUpSpecMgmtFecUnCorrectThres2.25 = 1
ccsUpSpecMgmtFecUnCorrectThres2.26 = 1
ccsUpSpecMgmtFecUnCorrectThres2.27 = 1
...
ccsUpSpecMgmtSnrPollPeriod.24 = 15
ccsUpSpecMgmtSnrPollPeriod.25 = 15
ccsUpSpecMgmtSnrPollPeriod.26 = 15
ccsUpSpecMgmtSnrPollPeriod.27 = 15
...
ccsUpSpecMgmtHopCondition.24 = snr(0)
ccsUpSpecMgmtHopCondition.25 = snr(0)
ccsUpSpecMgmtHopCondition.26 = snr(0)
ccsUpSpecMgmtHopCondition.27 = snr(0)
...
ccsUpSpecMgmtFromCenterFreq.24 = 10000
ccsUpSpecMgmtFromCenterFreq.25 = 15008
ccsUpSpecMgmtFromCenterFreq.26 = 20000
ccsUpSpecMgmtFromCenterFreq.27 = 25008
...
ccsUpSpecMgmtToCenterFreq.24 = 10000
ccsUpSpecMgmtToCenterFreq.25 = 15008
ccsUpSpecMgmtToCenterFreq.26 = 20000
```

```

ccsUpSpecMgmtToCenterFreq.27 = 25008
...
ccsUpSpecMgmtFromBandWidth.24 = 1600
ccsUpSpecMgmtFromBandWidth.25 = 3200
ccsUpSpecMgmtFromBandWidth.26 = 3200
ccsUpSpecMgmtFromBandWidth.27 = 3200
...
ccsUpSpecMgmtToBandWidth.24 = 1600
ccsUpSpecMgmtToBandWidth.25 = 3200
ccsUpSpecMgmtToBandWidth.26 = 3200
ccsUpSpecMgmtToBandWidth.27 = 3200
...
ccsUpSpecMgmtFromModProfile.24 = 1
ccsUpSpecMgmtFromModProfile.25 = 1
ccsUpSpecMgmtFromModProfile.26 = 1
ccsUpSpecMgmtFromModProfile.27 = 1
...
ccsUpSpecMgmtToModProfile.24 = 2
ccsUpSpecMgmtToModProfile.25 = 2
ccsUpSpecMgmtToModProfile.26 = 2
ccsUpSpecMgmtToModProfile.27 = 2
...
ccsUpSpecMgmtSNR.24 = 0
ccsUpSpecMgmtSNR.25 = 0
ccsUpSpecMgmtSNR.26 = 0
ccsUpSpecMgmtSNR.27 = 0

csh%

```

Using Flap Lists

To configure, clear, and access the flap lists on a Cisco CMTS router, use the following procedures:

- [Configuring Flap List Operation Using SNMP, page A-29](#)
- [Displaying the Flap List Using SNMP, page A-30](#)
- [Displaying Flap-List Information for Specific Cable Modems, page A-32](#)
- [Clearing the Flap List and Counters Using SNMP, page A-33](#)

Configuring Flap List Operation Using SNMP

To configure the Flap List Troubleshooting feature on the Cisco CMTS using SNMP, set the appropriate `cssFlapObjects` attributes in the CISCO-CABLE-SPECTRUM-MIB. [Table A-9](#) lists Flap-list configuration attributes:

Table A-9 Flap-List Configuration Attributes

| Attribute | Type | Range | Description |
|-------------------------------------|-----------|-------------------------|--|
| <code>cssFlapListMaxSize</code> | Integer32 | 1 to 65536 ¹ | The maximum number of modems that a flap list can support. The default is 100. |
| <code>cssFlapListCurrentSize</code> | Integer32 | 1 to 65536 ¹ | The current number of modems in the flap list. |

Table A-9 Flap-List Configuration Attributes (continued)

| Attribute | Type | Range | Description |
|-----------------------------|------------|-------------|---|
| ccsFlapAging | Integer32 | 1 to 86400 | The flap entry aging threshold in minutes. The default is 10080 minutes (180 hours or 7 days). |
| ccsFlapInsertionTime | Integer32 | 60 to 86400 | The worst-case insertion time, in seconds. If a cable modem has not completed the registration stage within this interval, the cable modem is inserted into the flap list. The default value is 90 seconds. |
| ccsFlapPowerAdjustThreshold | Integer32 | 1 to 10 | When the power of the modem is adjusted beyond the power adjust threshold, the modem is inserted into the flap list. |
| ccsFlapMissThreshold | Unsigned32 | 1 to 12 | When a cable modem does not acknowledge this number of consecutive MAC-layer station maintenance (keepalive) messages, the cable modem is placed in the flap list. |

1. The allowable range when using SNMP for these parameters is 1 to 65536 (a 32-bit value), but the valid operational range is 1 to 8191.

Displaying the Flap List Using SNMP

To display the contents of the flap list using SNMP, query the `cssFlapTable` in the `CISCO-CABLE-SPECTRUM-MIB`. This table contains an entry for each cable modem. [Table A-10](#) briefly describes each `cssFlapTable` attribute.

Table A-10 `cssFlapTable` Attributes

| Attribute | Type | Description |
|--------------------------|----------------|--|
| cssFlapMacAddr | MacAddress | MAC address of the cable modem's cable interface. Identifies a flap-list entry for a flapping cable modem. |
| ccsFlapUpstreamIfIndex | InterfaceIndex | Upstream being used by the flapping cable modem. |
| ccsFlapDownstreamIfIndex | InterfaceIndex | Downstream being used by the flapping cable modem. |
| ccsFlapLastFlapTime | DateAndTime | Time stamp for the last time the cable modem flapped. |
| ccsFlapCreateTime | DateAndTime | Time stamp that this entry was added to the table. |
| ccsFlapRowStatus | RowStatus | Control attribute for the status of this entry. |
| ccsFlapInsertionFailNum | Unsigned32 | Number of times the cable modem comes up and inserts itself into the network. This counter is increased when the time between initial link establishment and a reestablishment was less than the threshold parameter configured using the cable flap-list insertion-time command or <code>ccsFlapInsertionTime</code> attribute. When the cable modem cannot finish registration within the insertion time (<code>ccsFlapInsertionTime</code>), it resends the initial maintenance packet. When the CMTS receives the packet sooner than expected, the CMTS increments this counter. |
| ccsFlapHitNum | Unsigned32 | Number of times the CM responds to MAC-layer station maintenance (keepalive) messages. (The minimum hit rate is once per 30 seconds.) |

Table A-10 *ccsFlapTable Attributes (continued)*

| Attribute | Type | Description |
|---------------------------|-------------|---|
| ccsFlapMissNum | Unsigned32 | Number of times the CM misses and does not respond to a MAC-layer station maintenance (keepalive) message. An 8 percent miss rate is normal for the Cisco cable interface line cards. If the CMTS misses a ranging request within 25 msec, then the miss number is incremented. |
| ccsFlapCrcErrorNum | Unsigned32 | Number of times the CMTS upstream receiver flagged a packet with a CRC error. A high value indicates that the cable upstream may have a high noise level. The modem may not be flapping yet, but this could become a problem. |
| ccsFlapPowerAdjustmentNum | Unsigned32 | Number of times the cable modem upstream transmit power is adjusted during station maintenance. When the adjustment is greater than the power-adjustment threshold, the number is incremented. |
| ccsFlapTotalNum | Unsigned32 | Number of times a modem has flapped, which is the sum of the following: <ul style="list-style-type: none"> • When ccsFlapInsertionFailNum is increased • When the CMTS receives a miss followed by a hit • When ccsFlapPowerAdjustmentNum is increased |
| ccsFlapResetNow | Boolean | Setting this object to True (1) resets all flap-list counters to zero. |
| ccsFlapLastResetTime | DateAndTime | Time stamp for when all the counters for this particular entry were reset to zero. |

For example, the following shows the output for `ccsFlapTable` when it contains entries for two cable modems with the MAC addresses of 00.07.0E.02.CA.91 (0.7.14.2.202.145) and 00.07.0E.03.68.89 (0.7.14.3.104.137):

```

csh% getmany -v2c 10.10.11.12 public ccsFlapTable

ccsFlapUpstreamIfIndex.0.7.14.2.202.145 = 17
ccsFlapUpstreamIfIndex.0.7.14.3.104.137 = 17
ccsFlapDownstreamIfIndex.0.7.14.2.202.145 = 21
ccsFlapDownstreamIfIndex.0.7.14.3.104.137 = 21
ccsFlapInsertionFails.0.7.14.2.202.145 = 2
ccsFlapInsertionFails.0.7.14.3.104.137 = 0
ccsFlapHits.0.7.14.2.202.145 = 54098
ccsFlapHits.0.7.14.3.104.137 = 54196
ccsFlapMisses.0.7.14.2.202.145 = 65
ccsFlapMisses.0.7.14.3.104.137 = 51
ccsFlapCrcErrors.0.7.14.2.202.145 = 0
ccsFlapCrcErrors.0.7.14.3.104.137 = 0
ccsFlapPowerAdjustments.0.7.14.2.202.145 = 0
ccsFlapPowerAdjustments.0.7.14.3.104.137 = 0
ccsFlapTotal.0.7.14.2.202.145 = 5
ccsFlapTotal.0.7.14.3.104.137 = 4
ccsFlapLastFlapTime.0.7.14.2.202.145 = 14 03 04 1e 07 35 10 00
ccsFlapLastFlapTime.0.7.14.3.104.137 = 14 03 04 1e 07 34 12 00
ccsFlapCreateTime.0.7.14.2.202.145 = 14 03 04 1e 07 00 2b 00
ccsFlapCreateTime.0.7.14.3.104.137 = 14 03 04 1e 07 00 2c 00
ccsFlapRowStatus.0.7.14.2.202.145 = 1
ccsFlapRowStatus.0.7.14.3.104.137 = 1
ccsFlapInsertionFailNum.0.7.14.2.202.145 = 2
ccsFlapInsertionFailNum.0.7.14.3.104.137 = 0
ccsFlapHitNum.0.7.14.2.202.145 = 54098

```

```

ccsFlapHitNum.0.7.14.3.104.137 = 54196
ccsFlapMissNum.0.7.14.2.202.145 = 65
ccsFlapMissNum.0.7.14.3.104.137 = 51
ccsFlapCrcErrorNum.0.7.14.2.202.145 = 0
ccsFlapCrcErrorNum.0.7.14.3.104.137 = 0
ccsFlapPowerAdjustmentNum.0.7.14.2.202.145 = 0
ccsFlapPowerAdjustmentNum.0.7.14.3.104.137 = 0
ccsFlapTotalNum.0.7.14.2.202.145 = 5
ccsFlapTotalNum.0.7.14.3.104.137 = 4
ccsFlapResetNow.0.7.14.2.202.145 = 2
ccsFlapResetNow.0.7.14.3.104.137 = 2
ccsFlapLastResetTime.0.7.14.2.202.145 = 14 03 04 1e 06 39 0c 00
ccsFlapLastResetTime.0.7.14.3.104.137 = 14 03 04 1e 06 39 1e 00

csh%

```

**Tip**

To collect both the flap list configuration parameters and the contents of the flap list, perform a GET request of `ccsFlapObjects`.

Displaying Flap-List Information for Specific Cable Modems

To use SNMP requests to display flap-list information for a specific cable modem, use the cable modem's MAC address as the index to retrieve entries from the `ccsFlapTable`. Use the following procedure to retrieve flap-list entries for a particular cable modem.

DETAILED STEPS

-
- Step 1** Convert the cable modem's MAC address into a dotted decimal string. For example, the MAC address 000C.64ff.eb95 would become 0.12.100.255.235.149.
- Step 2** Use the dotted decimal version of the MAC address as the instance for requesting information from the `ccsFlapTable`. For example, to retrieve the `ccsFlapHits`, `ccsFlapMisses`, and `ccsFlapPowerAdjustments` values for this cable modem, you would make an SNMP request for the following objects:
- `ccsFlapHits.0.12.100.255.235.149`
 - `ccsFlapMisses.0.12.100.255.235.149`
 - `ccsFlapPowerAdjustments.0.12.100.255.235.149`
-

Example

Assume that you want to retrieve the same flap-list information as the **show cable flap-list** command for a cable modem with the MAC address of 000C.64ff.eb95:

```
Router# show cable flap-list
```

```

MAC Address      Upstream      Ins  Hit  Miss  CRC   P-Adj  Flap  Time
000C.64ff.eb95  Cable3/0/U4  3314 55605 50460 0     *42175 47533 Jan 27 02:49:10

```

```
Router#
```

Use an SNMP tool to retrieve the `ccsFlapTable` and filter it by the decimal MAC address. For example, using the standard Unix **getone** command, you would give the following command:

```
csh% getmany -v2c 192.168.100.121 public ccsFlapTable | grep 0.12.100.255.235.149
```

```

ccsFlapUpstreamIfIndex.0.12.100.255.235.149 = 15
ccsFlapDownstreamIfIndex.0.12.100.255.235.149 = 17
ccsFlapInsertionFails.0.12.100.255.235.149 = 3315
ccsFlapHits.0.12.100.255.235.149 = 55608
ccsFlapMisses.0.12.100.255.235.149 = 50460
ccsFlapCrcErrors.0.12.100.255.235.149 = 0
ccsFlapPowerAdjustments.0.12.100.255.235.149 = 42175
ccsFlapTotal.0.12.100.255.235.149 = 47534
ccsFlapLastFlapTime.0.12.100.255.235.149 = 07 d4 01 1b 02 33 1a 00
ccsFlapCreateTime.0.12.100.255.235.149 = 07 d4 01 16 03 23 22 00
ccsFlapRowStatus.0.12.100.255.235.149 = active(1)
ccsFlapInsertionFailNum.0.12.100.255.235.149 = 3315
ccsFlapHitNum.0.12.100.255.235.149 = 55608
ccsFlapMissNum.0.12.100.255.235.149 = 50460
ccsFlapCrcErrorNum.0.12.100.255.235.149 = 0
ccsFlapPowerAdjustmentNum.0.12.100.255.235.149 = 42175
ccsFlapTotalNum.0.12.100.255.235.149 = 47534
ccsFlapResetNow.0.12.100.255.235.149 = false(2)
ccsFlapLastResetTime.0.12.100.255.235.149 = 07 d4 01 16 03 20 18 00

csh%

```

To request just one particular value, use the decimal MAC address as the instance for that object:

```

csh% getone -v2c 172.22.85.7 public ccsFlapMisses.0.12.100.255.235.149

ccsFlapMisses.0.12.100.255.235.149 = 50736

csh %

```

Clearing the Flap List and Counters Using SNMP

To remove a cable modem from the flap list or to clear one or all of the flap-list counters, set the appropriate `ccsFlapObjects` attributes in the CISCO-CABLE-SPECTRUM-MIB. [Table A-11](#) lists the attributes that clear the SNMP counters.

Table A-11 Attributes to Clear the Flap List

| Attribute | Type | Description |
|------------------------------|---------|---|
| <code>ccsFlapResetAll</code> | Boolean | Setting this object to True (1) resets all flap-list counters to zero. |
| <code>ccsFlapClearAll</code> | Boolean | Setting this object to True (1) removes all cable modems from the flap list, and destroys all entries in the <code>ccsFlapTable</code> . If a modem keeps flapping, the modem is added again into the flap list as a new entry. |



Note

The `ccsFlapLastClearTime` attribute contains the date and time that the entries in the `ccsFlapTable` table were last cleared.

Using Subscriber Traffic Management

You can use the attributes in the CISCO-CABLE-QOS-MONITOR-MIB to configure subscriber traffic management on the Cisco CMTS router, and to display the cable modems that have violated their configured bandwidth limits. Use the following procedures to perform these tasks:

- [Configuring and Displaying the Enforce Rules for Quality of Service, page A-34](#)
- [Displaying Subscribers Who Have Violated Their Enforce Rules, page A-35](#)
- [Notifications for Subscribers Who Have Violated Their Enforce Rules, page A-36](#)

Configuring and Displaying the Enforce Rules for Quality of Service

Use the attributes in the ccqmCmtsEnforceRuleTable table to configure the QoS enforce rules that should be enforced on the Cisco CMTS router. You can also query this table to display the enforce rules that are currently defined.

[Table A-12](#) lists the attributes that are in each row of the ccqmCmtsEnforceRuleTable table. One row entry (CcqmCmtsEnforceRuleEntry) exists for each enforce rule and is indexed by the unique rule name (ccqmCmtsEnfRuleName).

Table A-12 Attributes for the QoS Enforce Rules

| Attribute | Type | Description |
|------------------------------|---------------|--|
| ccqmCmtsEnfRuleName | DisplayString | Unique name of the enforce rule, up to 15 characters long. |
| ccqmCmtsEnfRuleRegQoS | Unsigned32 | If not set to 0, this attribute is an index into the docsIfQosProfileTable table to identify the QoS profile to be used as the registered profile with this rule. |
| ccqmCmtsEnfRuleEnfQoS | Unsigned32 | If not set to 0, this attribute is an index into the docsIfQosProfileTable table to identify the QoS profile to be used as the enforced profile that should be put into effect when subscribers violate their bandwidth limits. |
| ccqmCmtsEnfRuleMonDuration | Unsigned32 | Time period, in minutes, of the monitoring window over which the users are monitored to determine whether they are violating their bandwidth limits. The valid range is 10 to 10,080 minutes (7 days), with a default value of 360 minutes (6 hours). |
| ccqmCmtsEnfRuleSampleRate | Unsigned32 | How often, in minutes, the Cisco CMTS checks the subscriber's bandwidth usage to determine whether they are violating their bandwidth limits. The valid range is 10 to 120 minutes (2 hours), with a default of 15 minutes. |
| ccqmCmtsEnfRulePenaltyPeriod | Unsigned32 | How long, in minutes, the subscriber remains in the penalty period after violating their bandwidth limits. The subscriber is forced to use the enforced QoS profile until this time period expires or until the penalty period is manually cleared on the Cisco CMTS. The valid range is 1 to 10,080 minutes, with a default of 10,080 (7 days). |

Table A-12 *Attributes for the QoS Enforce Rules (continued)*

| Attribute | Type | Description |
|----------------------------|-------------------|--|
| ccqmCmtsEnfRuleByteCount | Unsigned32 | Maximum number of kilobytes that the subscriber can transmit or receive (depending on the enforce rule's direction) during each monitoring window time period. If the subscriber exceeds this number of bytes, they are flagged as overconsuming and possibly subject to the penalty period. The valid range is any 32-bit integer value, with no default. |
| ccqmCmtsEnfRuleDirection | CCQMRuleDirection | Specifies the direction (upstream, downstream, or bidirectional) over which bytes are monitored for this enforce rule. |
| ccqmCmtsEnfRuleAutoEnforce | TruthValue | Specifies whether the enforced QoS profile is automatically applied when the subscriber violates the bandwidth limits. The default is False (the enforced QoS profile is not automatically applied). |
| ccqmCmtsEnfRuleRowStatus | RowStatus | Specifies the status of this particular row. This enforce rule entry becomes active only when the RowStatus is set to active(1). To change any of the row parameters, however, the RowStatus must first be set to notInService(2), which ends the current monitoring of users with this enforce rule. Then the row must be set back to active(1) before monitoring starts again. |

Displaying Subscribers Who Have Violated Their Enforce Rules

Table A-13 lists the attributes available for each entry (ccqmEnfRuleViolateEntry) in the ccqmEnfRuleViolateTable table, which lists the subscribers who have violated their enforce rule over the configured monitoring time period.

Table A-13 *Attributes for ccqmEnfRuleViolateTable Table*

| Attribute | Type | Description |
|----------------------------------|---------------|--|
| ccqmEnfRuleViolateID | Unsigned32 | Unique ID assigned to this row entry. For DOCSIS 1.0 users, this is the same value as the subscriber's service flow ID (SFID). |
| ccqmEnfRuleViolateMacAddr | MacAddress | MAC (hardware) address of the cable interface for the subscriber who has violated the QoS bandwidth limits. |
| ccqmEnfRuleViolateRuleName | DisplayString | Name of the enforce rule that is associated with this subscriber. This value can be compared to the ccqmCmtsEnfRuleName attribute to find the enforce rule assigned to this subscriber. |
| ccqmEnfRuleViolateByteCount | Unsigned32 | Total number of kilobytes that the subscriber used during the monitoring time period. (This counter is reset whenever the enforce rule is stopped and restarted.) |
| ccqmEnfRuleViolateLastDetectTime | DateAndTime | The time stamp for when the Cisco CMTS determined that the subscriber had violated the enforce rule. |
| ccqmEnfRuleViolatePenaltyExpTime | DateAndTime | The time stamp for when the penalty period expires for this subscriber. Unless the penalty period is manually cancelled by an operator at the Cisco CMTS router, the subscriber remains in the penalty period until this date and time, at which point their original QoS profile is restored. This attribute is 0 when the subscriber's enforce rule did not include an enforced QoS Profile. |

Notifications for Subscribers Who Have Violated Their Enforce Rules

To configure the Cisco CMTS router so that it sends a notification whenever a user violates the bandwidth limits in the enforce rules, set the `ccqmEnfRuleViolateNotifEnable` object to True. The default is False (notifications are not sent). [Table A-14](#) lists the attributes (which are defined in the `ccqmEnfRuleViolateTable` table) that are sent in each notification.

Table A-14 Attributes for `ccqmEnfRuleViolateNotification`

| Attribute | Type | Description |
|---|---------------|--|
| <code>ccqmEnfRuleViolateMacAddr</code> | MacAddress | MAC (hardware) address of the cable interface for the subscriber that has violated the QoS bandwidth limits. |
| <code>ccqmEnfRuleViolateRuleName</code> | DisplayString | Name of the enforce rule that is associated with this subscriber. This value can be compared to the <code>ccqmCmtsEnfRuleName</code> attribute to find the enforce rule assigned to this subscriber. |
| <code>ccqmEnfRuleViolatePenaltyExpTime</code> | DateAndTime | The time stamp for when the penalty period expires for this subscriber. Unless the penalty period is manually cancelled by an operator at the Cisco CMTS router, the subscriber remains in the penalty period until this date and time, at which point their original QoS profile is restored. This attribute is 0 when the subscriber's enforce rule did not include an enforced QoS Profile. |
| <code>ccqmEnfRuleViolateByteCount</code> | Unsigned32 | Total number of kilobytes that the subscriber used during the monitoring time period. (This counter is reset whenever the enforce rule is stopped and restarted.) |

Usage-Based Billing

The CISCO-CABLE-METERING-MIB allows for configuration of the parameters that control the metering record collection like interval, amount of metering information required, location or IP address of the collection server, and metering filename. In addition it provides for some important notifications to the NMS to indicate the success or failure of the metering collection.

Usage-Based Billing feature for the Cisco Cable Modem Termination System (CMTS), which provides subscriber account and billing information in the Subscriber Account Management Interface Specification (SAMIS) format. The SAMIS format is specified by the Data-over-Cable Service Interface Specifications (DOCSIS) Operations Support System Interface (OSSI) specification.

The Usage-Based Billing feature provides a standards-based, open application approach to recording and retrieving traffic billing information for DOCSIS networks. When enabled, this feature provides the following billing information about the cable modems and customer premises equipment (CPE) devices that are using the cable network:

- IP and MAC addresses of the cable modem.
- Service flows being used (both upstream and downstream service flows are tracked).
- IP addresses for the CPE devices that are using the cable modem.
- Total number of octets and packets received by the cable modem (downstream) or transmitted by the cable modem (upstream) during the collection period.

- Total number of downstream packets for the cable modem that the CMTS dropped or delayed because they would have exceeded the bandwidth levels allowed by the subscriber's service level agreement (SLA).

Billing records are maintained in a standardized text format that the service provider can easily integrate into their existing billing applications. Service providers can use this information to determine which users might be potential customers for service upgrades, as well as those customers that might be trying to exceed their SLA limits on a regular basis.

Modes of Operation

The Usage-Based Billing feature operates in two modes:

- File Mode
- Streaming Mode

File Mode

In file mode, the CMTS collects the billing record information and writes the billing records to a file on a local file system, using a file name that consists of the router's hostname followed by a timestamp of when the file was written. A remote application can then log into the CMTS and transfer the billing record file to an external server where the billing application can access it.

The remote application can use the Secure Copy Protocol (SCP) or the Trivial File Transfer Protocol (TFTP) to transfer the file. After a successful transfer, the remote application then deletes the billing record file, which signals the CMTS that it can create a new file. The remote application can either periodically log into the CMTS to transfer the billing record file, or it can wait until the CMTS sends an SNMPv3 trap to notify the application that a billing record file is available.

Usage-Based Billing Feature (File Mode)

To configure the Cisco CMTS for Usage-Based Billing feature in file mode, you must set a number of objects in the CISCO-CABLE-METERING-MIB. [Table A-15](#) describes CISCO-CABLE-METERING-MIB objects to be configured for file mode.

Table A-15 CISCO-CABLE-METERING-MIB Objects to be Configured for File Mode

| MIB Object | Type | Description |
|----------------------|---------|---|
| ccmtrCollectionTable | | |
| ccmtrCollectionType | Integer | <p>Enables or disables the Usage-Based Billing feature. The valid values are:</p> <ul style="list-style-type: none"> • 1 = none. The Usage-Based Billing feature is disabled (default). • 2 = local. The Usage-Based Billing feature is enabled and configured for file mode. • 3 = stream. The Usage-Based Billing feature is enabled and configured for streaming mode. <p>Set ccmCollectionType to 2 (local) to enable the feature for file mode.</p> |

Table A-15 CISCO-CABLE-METERING-MIB Objects to be Configured for File Mode (continued)

| MIB Object | Type | Description |
|---------------------------|---------------|---|
| ccmtrCollectionFilesystem | DisplayString | <p>Specifies the file system where the billing record file should be written. This object has a maximum length of 25 characters and must specify a valid file system on the router (such as slot0, disk1, or flash).</p> <p>Note The Cisco CMTS writes the billing records to this file system using a file name that consists of the router's hostname followed by a timestamp when the record was written.</p> |
| ccmtrCollectionCpeList | TruthValue | <p>(Optional) Indicates whether IP addresses for customer premises equipment (CPE) devices are omitted from the billing records, so as to reduce the size of the billing records and to improve performance. The valid values are the following:</p> <ul style="list-style-type: none"> • true = CPE information is present (default). • false = CPE information is omitted. <p>Note When set to true, a maximum of 5 CPE IP addresses for each cable modem.</p> |
| ccmtrCollectionAggregate | TruthValue | <p>(Optional) Indicates whether all information for an individual cable modem is combined into one record. Separate counters are maintained for upstream and downstream traffic, but those counters include all service flows in that direction. The valid values are as follows:</p> <ul style="list-style-type: none"> • true = All service flow information for each cable modem is aggregated into a single billing record. In this configuration, the service flow ID (SFID) for the billing record is set to 0 and the service class name (SCN) is blank. • false = Information for each cable modem is not aggregated into a single billing record, but instead each service flow is recorded into its own record (default). |
| ccmtrCollectionSrcIfIndex | | <p>Indicates the source interface for the billing packets. If the ccmtrCollectionType is local(2), the IP address of this interface is used as the CMTS IP address in the billing packets. If the ccmtrCollectionType is remote(3), the source IP address of the billing packets as well as the CMTS IP address in billing packets is changed to the IP address of this interface.</p> <p>Note that metering mode must be configured to specify the source-interface for metering.</p> |

Streaming Mode

In streaming mode, the CMTS collects the billing record information and then regularly transmits the billing record file to an application on an external server, using either a non-secure TCP connection or a secure sockets layer (SSL) connection. When the external server acknowledges a successful transfer, the CMTS deletes the billing record file and begins creating a new file.

If the CMTS fails to establish a successful connection with the external server, it retries the connection between 1 and 3 times, depending on the configuration. If the CMTS continues to be unable to connect with the external server, the CMTS can send an SNMPv3 trap to notify an SNMP manager that this failure occurred.

In streaming mode, you configure the CMTS to transmit the billing record file at regular intervals. Typically, the interval chosen would depend on the number of cable modems and the size of the billing record files that the CMTS produces.

To configure the Cisco CMTS for Usage-Based Billing feature in streaming mode, you must set a number of objects in the CISCO-CABLE-METERING-MIB. [Table A-16](#) describes CISCO-CABLE-METERING-MIB objects to be configured for streaming mode of these objects.

Table A-16 CISCO-CABLE-METERING-MIB Objects to be Configured for Streaming Mode

| Object | Type | Description |
|-----------------------------|-----------------|--|
| ccmtrCollectionTable | | |
| ccmtrCollectionType | Integer | Enables or disables the Usage-Based Billing feature. The valid values are: <ul style="list-style-type: none"> • none(1)–The Usage-Based Billing feature is disabled (default). • local(2)–The Usage-Based Billing feature is enabled and configured for file mode. • stream(3)–The Usage-Based Billing feature is enabled and configured for streaming mode. Set ccmCollectionType to stream(3) to enable the feature for streaming mode. |
| ccmtrCollectionIpAddress | InetAddress | IP address for the external collection server. This value must be specified. |
| ccmtrCollectionPort | Unsigned32 | TCP port number at the external collection server to which the billing records should be sent. The valid range is from 0 to 65535, but you should not specify a port in the well-known range from 0 to 1024. This value must be specified. |
| Note | | You can configure the ccmCollectionIpAddress and ccmCollectionPort objects twice, to specify a primary collection server and a secondary collection server. |
| ccmtrCollectionIpAddrType | InetAddressType | (Optional) Type of IP address being used for the collection server. The only valid value is ipv4, which is the default value. |
| ccmCollectionInterval | Unsigned32 | (Optional) Specifies how often, in minutes, the billing records are streamed to the external server. The valid range is from 15 to 1440 minutes (24 hours), with a default of 30 minutes. (Cisco recommends a minimum interval of 30 minutes.) |
| ccmtrCollectionRetries | Unsigned32 | (Optional) Specifies the number of retry attempts that the CMTS makes to establish a secure connection with the external server before using the secondary server (if configured) and sending an SNMP trap about the failure. The valid range for <i>n</i> is from 0 to 5, with a default of 0. |

Note The ccmCollectionInterval and ccmCollectionRetries parameters are optional when configuring Usage-Based Billing for streaming mode with SNMP commands, but these parameters are required when configuring the feature with CLI commands.

Table A-16 CISCO-CABLE-METERING-MIB Objects to be Configured for Streaming Mode (continued)

| Object | Type | Description |
|--------------------------|------------|---|
| ccmtrCollectionSecure | TruthValue | <p>(Optional) Specifies whether the Cisco CMTS should use a secure socket layer (SSL) connection when connecting with the billing application on the external server. The valid values are:</p> <ul style="list-style-type: none"> • true(1)—The Cisco CMTS uses a SSL connection. This option is available only on CMTS software images that support Baseline Privacy Interface (BPI) encryption. • false(2)—The Cisco CMTS uses an unencrypted TCP connection. This is the default value. |
| ccmtrCollectionCpeList | TruthValue | <p>(Optional) Indicates whether IP addresses for customer premises equipment (CPE) devices are omitted from the billing records, so as to reduce the size of the billing records and to improve performance. The valid values are the following:</p> <ul style="list-style-type: none"> • true—CPE information is present (default). • false—CPE information is omitted. <p>Note When set to true, a maximum of 5 CPE IP addresses for each cable modem.</p> |
| ccmtrCollectionAggregate | TruthValue | <p>(Optional) Indicates whether all information for an individual cable modem is combined into one record. Separate counters are maintained for upstream and downstream traffic, but those counters include all service flows in that direction. The valid values are as follows:</p> <ul style="list-style-type: none"> • true—All service flow information for each cable modem is aggregated into a single billing record. In this configuration, the service flow ID (SFID) for the billing record is set to 0 and the service class name (SCN) is blank. • false—Information for each cable modem is not aggregated into a single billing record, but instead each service flow is recorded into its own record (default). |

For complete documentation about using the Usage-Based Billing feature for Cisco CMTS, go to:

<http://www.cisco.com/en/US/docs/cable/cmts/feature/ubrsamis.html>

Identifying Cisco Unique Device Identifiers

In order to use UDI retrieval, the Cisco product in use must be UDI-enabled. A UDI-enabled Cisco product supports five required Entity MIB objects. The five Entity MIB v2 (RFC-2737) objects are as follows:

- entPhysicalName
- entPhysicalDescr
- entPhysicalModelName
- entPhysicalHardwareRev

- entPhysicalSerialNum

Although the show inventory command may be available, using that command on devices that are not UDI-enabled produces no output.

Before using the UDI Retrieval feature, you should understand the following concepts:

- Unique Device Identifier Overview—Each identifiable product is an entity, as defined by the Entity MIB (RFC-2737) and its supporting documents. Some entities, such as a chassis, have subtentities like slots. An Ethernet switch might be a member of a superentity like a stack. Most Cisco entities that are orderable products leaves the factory with an assigned UDI.
- The UDI information is printed on a label that is affixed to the physical hardware device, and it is also stored electronically on the device in order to facilitate remote retrieval.

A UDI consists of the following elements:

- Product identifier (PID)—The PID is the name by which the product can be ordered; it has been historically called the Product Name or Part Number. This is the identifier that one would use to order an exact replacement part.
- Version identifier (VID)—The VID is the version of the product. Whenever a product has been revised, the VID is incremented. The VID is incremented according to a rigorous process derived from Telcordia GR-209-CORE, an industry guideline that governs product change notices.
- Serial number (SN)—The SN is the vendor-unique serialization of the product. Each manufactured product carries a unique serial number assigned at the factory, which cannot be changed in the field. This is the means by which to identify an individual, specific instance of a product.

DOCS-DSG-IF-MIB Validation Requirements

Validation describes mandatory parameters that must be set and consistency rules so that if the data is set, the data must also conform to those rules. This section describes the implementation relative to the DOCS-DSG-IF-MIB. The validation rules are:

- All indices of MIB tables in DOCS-DSG-IF-MIB must be within the range from 1 to 65535 in order to keep index range consistent between SNMP and CLI to ensure inter-operability.

The valid values of a RowStatus column are:

- active—row is complete and usable.
- notInService—row is complete but not active.
- notReady—the row is missing columns required for it to be set active.
- createAndGo—create the row and have it go active in a single Set request.
- createAndWait—create the row but keep it in notInService or notReady state until activated.
- destroy—remove the row.

Table A-17 lists the capabilities of each MIB table in the DOCS-DSG-IF-MIB.

Table A-17 DOCS-DSG-IF-MIB Table Capabilities

| MIB Table/Object | Maximum Access | Capability |
|------------------------------|-----------------------|--|
| dsgIfClassifierTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • CreateAndWait • Can be modified • Destroyed |
| dsgIfTunnelTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • CreateAndWait • Can be modified • Destroyed |
| dsgIfTunnelGrpToChannelTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • CreateAndWait • Can be modified • Destroyed |
| dsgIfDownstreamTable | Read-Write | Can be modified. |
| dsgIfClientIdTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • Destroyed |
| dsgIfVendorParamTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • Destroyed |
| dsgIfChannelListTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • Destroyed |
| dsgIfTimerTable | Read-Create | Values: <ul style="list-style-type: none"> • CreateAndGo • Destroyed |