APPENDIX A

Using MIBs

This chapter describes how to use SNMP to perform tasks on the Cisco 10000 Series. For information about how to avoid performance problems when you use SNMP to poll the router for routing table entries, see the "Considerations for Working with MIBs" section on page 2-1.

- Managing Physical Entities, page A-1
- Using Alarms to Monitor Outages, page A-11
  - Viewing Active Alarms Through the CLI, page A-12
  - Using the CISCO-ENTITY-ALARM-MIB to Monitor Alarms, page A-12
  - Enabling Traps and Syslog Messages for Alarms, page A-16
- Monitoring Router Interfaces, page A-16
  - Enabling Interface linkUp/linkDown Traps, page A-17
  - SNMP Trap Filtering for linkDown Traps, page A-18
- Monitoring PXF Utilization, page A-18
- Preprovisioning Line Cards, page A-20
- Replacing Line Cards—MIB State Characteristics, page A-21
- Performing Bulk-File Retrieval, page A-22
- Monitoring Quality of Service, page A-28
- Billing Customers for Traffic, page A-44
- Using CISCO-AAA-SESSION-MIB, page A-47
- Using CISCO-CBP-TARGET-MIB, page A-48
- Cisco Unique Device Identifier Support, page A-50

Managing Physical Entities

This section describes how to use SNMP to manage the physical entities (components) in the router by:

- Performing Inventory Management, page A-3
  - Determining the ifIndex Value for a Physical Port, page A-8
  - Tagging Router Assets, page A-8
- Monitoring and Configuring FRU Status, page A-8

- Generating SNMP Traps, page A-9

See the "Preprovisioning Line Cards" section on page A-20 for information about how to use SNMP to preconfigure the operating characteristics of a line card before the line card is inserted into the chassis.

Purpose and Benefits

The physical entity management feature of the Cisco 10000 SNMP implementation does the following:

- 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)
- Provides information about physical port to interface mappings
- Provides asset information for asset tagging
- Provides firmware and software information for chassis components

MIBs Used for Physical Entity Management

- 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, manufacturing assembly number, and 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.
- ENTITY-MIB—Contains information for managing physical entities on the router. It also organizes the entities into a containment tree that depicts their hierarchy and relationship to each other. The MIB contains the following tables:
  - The entPhysicalTable describes each physical component (entity) in the router. The table contains an entry for the top-level entity (the chassis) and for each entity in the chassis. Each entry provides information about that entity: its name, type, vendor, and a description, and describes how the entity fits into the hierarchy of chassis entities.

    Each entity is identified by a unique index (*entPhysicalIndex*) that is used to access information about the entity in this and other MIBs.
  - The entAliasMappingTable maps each physical port's entPhysicalIndex value to its corresponding ifIndex value in the IF-MIB ifTable.
  - The entPhysicalContainsTable shows the relationship between physical entities in the chassis. For each physical entity, the table lists the entPhysicalIndex for each of the entity's child objects.

Performing Inventory Management

Perform a MIB walk on the ENTITY-MIB entPhysicalTable to obtain information about entities in the router.

Figure A-1 through Figure A-5 show how entries in the entPhysicalTable provide information about entities.

Notes about entPhysicalTable Entries

As you examine entries in the ENTITY-MIB entPhysicalTable, consider the following:

- entPhysicalIndex—Uniquely identifies each entity in the chassis. This index is also used to access information about the entity in other MIBs.
- entPhysicalContainedIn—Indicates the entPhysicalIndex of a component's parent entity.
- entPhysicalParentRelPos—Shows the relative position of same-type entities that have the same entPhysicalContainedIn value (for example, chassis slots and line card ports). (See Figure A-5.)

Sample entPhysicalTable Entries

The figures in this section show how information is stored in the entPhysicalTable. You can determine the router configuration by examining entPhysicalTable entries.

Figure A-1 shows the ENTITY-MIB entPhysicalTable entries for a Gigabit Ethernet line card installed in slot 1 of the router chassis, and for the port on that line card.

Figure A-1 entPhysicalTable Entries for Chassis Entities

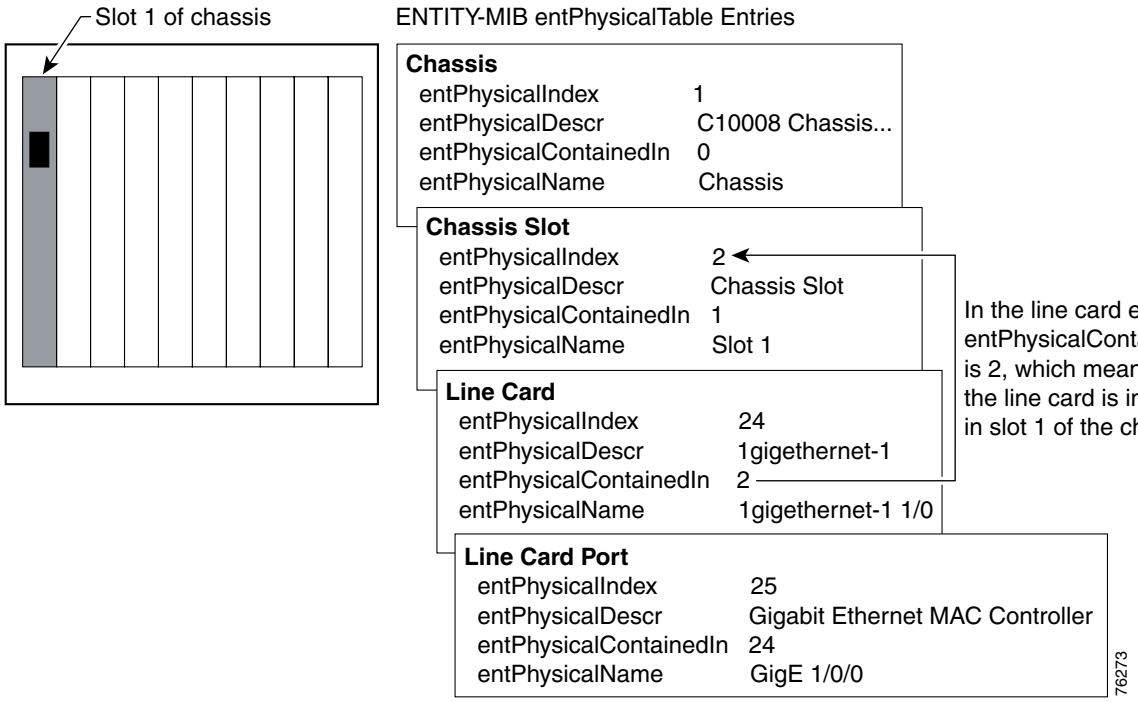


Figure A-2 shows sample entPhysicalTable entries for the entities shown in Figure A-4 and Figure A-5.

Figure A-2 Sample entPhysicalTable Entries

entPhysicalTable
entPhysicalEntry.*entPhysicalIndex*

entPhysicalEntry.1
 entPhysicalDescr C10008 chassis, Hw Serial#:...
 entPhysicalVendorType CiscoModules.3.1.3.303
 entPhysicalContainedIn 0
 entPhysicalClass chassis(3)
 entPhysicalParentRelPos -1

entPhysicalEntry.2
 entPhysicalDescr Chassis Slot
 entPhysicalVendorType CiscoModules.3.1.5.86
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 1
 entPhysicalName slot 1

entPhysicalEntry.3
 entPhysicalDescr Chassis Slot
 entPhysicalVendorType CiscoModules.3.1.5.86
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 2
 entPhysicalName slot 2

entPhysicalEntry.4
 entPhysicalDescr Chassis Slot
 entPhysicalVendorType CiscoModules.3.1.5.86
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 3
 entPhysicalName slot 3

entPhysicalEntry.5
 entPhysicalDescr Chassis Slot
 entPhysicalVendorType CiscoModules.3.1.5.86
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 4
 entPhysicalName slot 4

entPhysicalEntry.6
 entPhysicalDescr Chassis Slot
 entPhysicalVendorType CiscoModules.3.1.5.86
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 5
 entPhysicalName slot A
 . . .

entPhysicalContainedIn is the **entPhysicalIndex** of an entity's parent.

entPhysicalEntry.12
 entPhysicalDescr Power Supply Container
 entPhysicalVendorType CiscoModules.3.1.5.87
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 12

entPhysicalEntry.13
 entPhysicalDescr Power Supply
 entPhysicalVendorType CiscoModules.3.1.6.56
 entPhysicalContainedIn 12
 entPhysicalClass powerSupply(6)
 entPhysicalParentRelPos 1
 . . .
entPhysicalEntry.15
 entPhysicalDescr Fan Tray Container
 entPhysicalVendorType CiscoModules.3.1.5.88
 entPhysicalContainedIn 1
 entPhysicalClass container(5)
 entPhysicalParentRelPos 12

entPhysicalEntry.16
 entPhysicalDescr Fan Tray
 entPhysicalVendorType CiscoModules.3.1.7.25
 entPhysicalContainedIn 15
 entPhysicalClass module(9)
 entPhysicalParentRelPos 1
 . . .
entPhysicalEntry.20
 entPhysicalDescr Route Processor
 entPhysicalVendorType CiscoModules.3.1.9.5.29
 entPhysicalContainedIn 6
 entPhysicalClass module(9)
 entPhysicalParentRelPos 1

entPhysicalEntry.21
 entPhysicalDescr Forwarding Processor
 entPhysicalVendorType CiscoModules.3.1.9.5.30
 entPhysicalContainedIn 20
 entPhysicalClass module(9)
 entPhysicalParentRelPos 1

Figure A-3 Sample entPhysicalTable Entries (continued)

entPhysicalTable *(continued)*
entPhysicalEntry.*entPhysicalIndex*

entPhysicalEntry.24
entPhysicalDescr 1gigethernet-1
entPhysicalVendorType CiscoModules.3.1.9.32.3
entPhysicalContainedIn 2
entPhysicalClass module(9)
entPhysicalParentRelPos 1

entPhysicalEntry.25
entPhysicalDescr Gigabit Ethernet MAC Controller
entPhysicalVendorType CiscoModules.3.1.10.109
entPhysicalContainedIn 24
entPhysicalClass port(10)
entPhysicalParentRelPos 1
entPhysicalName GigE 1/0/0

entPhysicalEntry.26
entPhysicalDescr 6cht3-1
entPhysicalVendorType CiscoModules.3.1.9.32.2
entPhysicalContainedIn 3
entPhysicalClass module(9)
entPhysicalParentRelPos 1

entPhysicalEntry.27
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 1
entPhysicalName Serial2/0/0

entPhysicalEntry.28
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 2
entPhysicalName Serial2/0/1

entPhysicalEntry.29
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 3
entPhysicalName Serial2/0/2

entPhysicalEntry.30
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 4
entPhysicalName Serial2/0/3

entPhysicalEntry.31
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 5
entPhysicalName Serial2/0/4

entPhysicalEntry.32
entPhysicalDescr PMC FREEM, PMC S/UNI...
entPhysicalVendorType CiscoModules.3.1.10.20
entPhysicalContainedIn 26
entPhysicalClass port(10)
entPhysicalParentRelPos 6
entPhysicalName Serial2/0/5

entPhysicalEntry.33
entPhysicalDescr 1oc12pos-1
entPhysicalVendorType CiscoModules.3.1.9.32.1
entPhysicalContainedIn 4
entPhysicalClass module(9)
entPhysicalParentRelPos 1

entPhysicalEntry.34
entPhysicalDescr Skystone 4302 Sonet Framer
entPhysicalVendorType CiscoModules.3.1.10.52
entPhysicalContainedIn 33
entPhysicalClass port(10)
entPhysicalParentRelPos 1
entPhysicalName POS3/0/0

76275

Figure A-4 shows the entPhysicalTable entries for all of the line cards and line card ports in the configuration.

Figure A-4 entPhysicalTable Entries for Line Cards and Line Card Ports

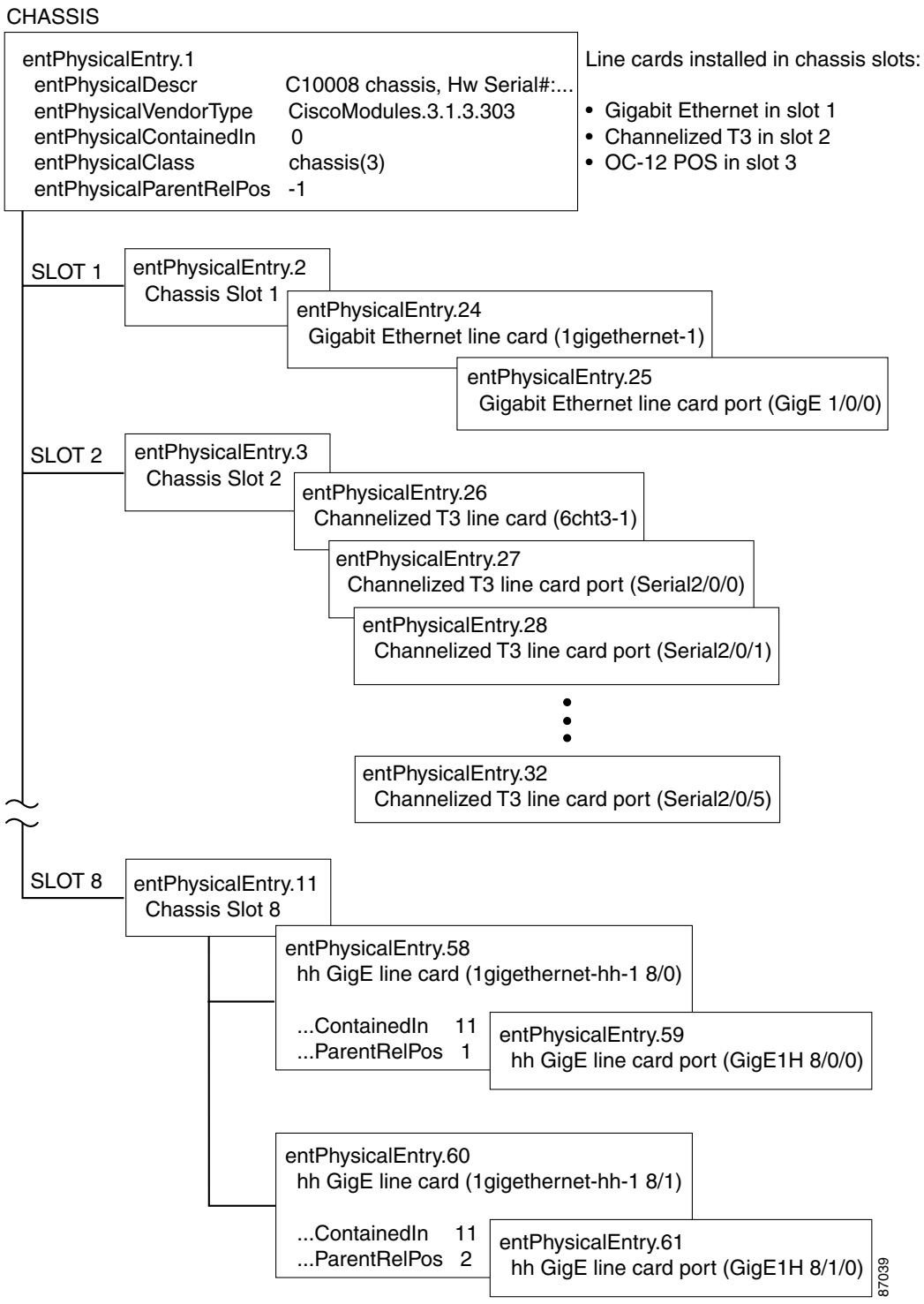
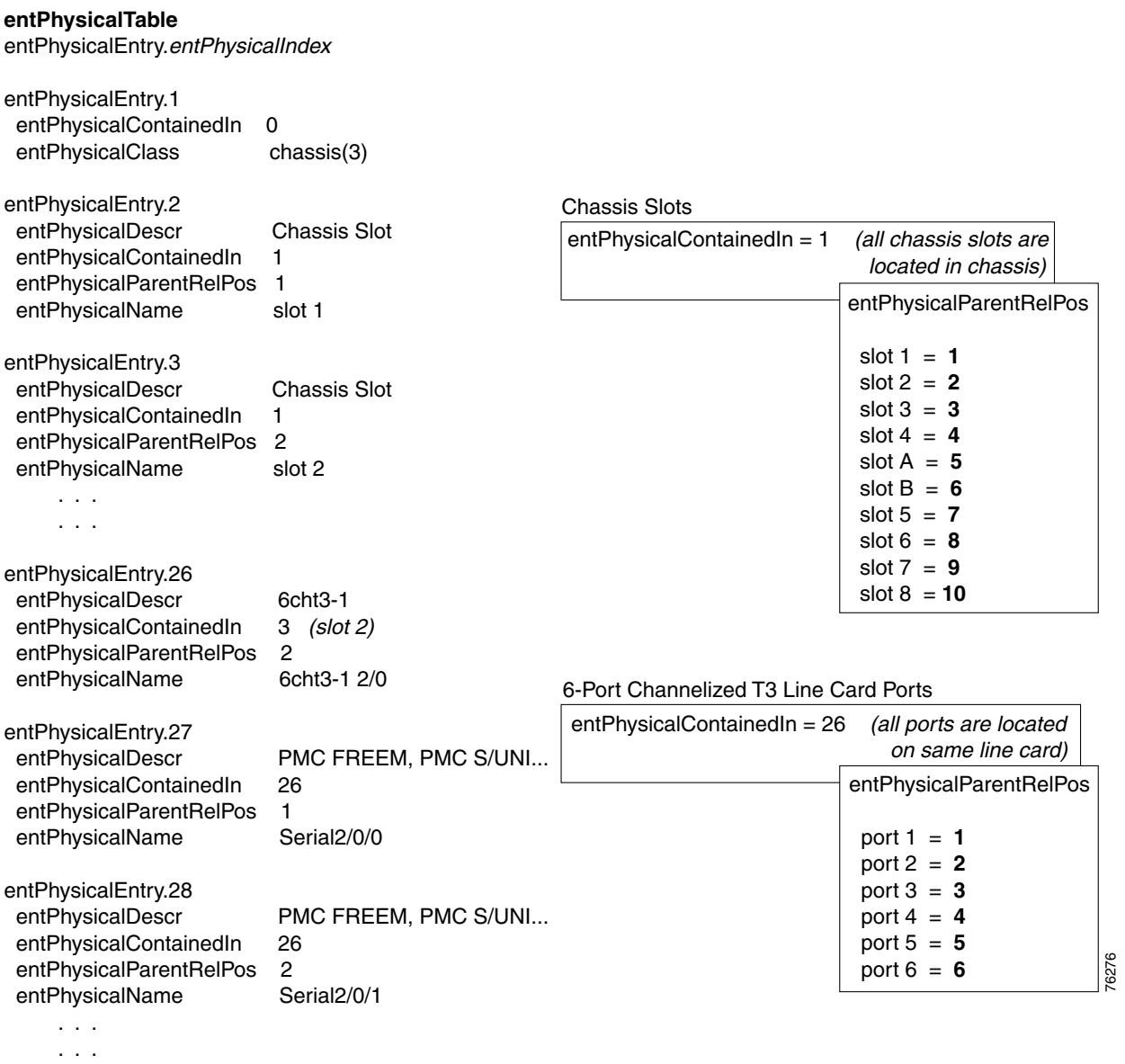


Figure A-5 shows how entPhysicalParentRelPos indicates the position of same-type entities within their parent object. Note that the entPhysicalTable entries on the left include relevant fields only.

Figure A-5 entPhysicalParentRelPos Values for Chassis Slots



Note the following about the sample configuration:

- All chassis slots and line card ports have the same entPhysicalContainedIn value:
  - For chassis slots, entPhysicalContainedIn = 1 (the entPhysicalIndex of the chassis).
  - For line card ports, entPhysicalContainedIn = 26 (the entPhysicalIndex of the line card).
- Each chassis slot and line card port has a different entPhysicalParentRelPos to show its relative position within the parent object.
- The 6-port channelized T3 line card is installed in slot 2 of the chassis.

Determining the ifIndex Value for a Physical Port

The ENTITY-MIB **entAliasMappingIdentifier** maps a physical port to an interface by mapping the port's entPhysicalIndex to its corresponding ifIndex value in the IF-MIB ifTable. For example, the following sample shows that the physical port whose entPhysicalIndex is 35 is associated with the interface whose ifIndex value is 4. (See the MIB for detailed descriptions of possible MIB values.)

```
entAliasMappingIdentifer.35.0 = ifIndex.4
```

Tagging Router Assets

You can use the CISCO-ENTITY-ASSET-MIB **ceAssetTag** object to assign a unique, nonvolatile identifier (tag) to any line card or Performance Routing Engine (PRE) that you want to keep track of. The tag remains with the PRE or line card even if it is installed in another chassis.

For example, the following ceAssetTable entry shows an asset tag of `pre-1-1ge` being assigned to the Gigabit Ethernet line card whose serial number is `CAB0430AXEU`. Even if the line card is removed from this chassis and installed in another chassis, its ceAssetTag remains `pre-1-1ge`.

```
ceSerialNumber = CAB0430AXEU
ceOrderablePartNumber = ESR-1GE
ceAssetTag = pre-1-1ge
  .  .  .
```

Monitoring and Configuring FRU Status

View objects in the CISCO-ENTITY-FRU-CONTROL-MIB cefcModuleTable to determine the administrative and operational status of FRUs, such as power supplies and line cards:

- cefcModuleAdminStatus—The administrative state of the FRU. Use cefcModuleAdminStatus to enable or disable the FRU.
- cefcModuleOperStatus—The current operational state of the FRU.

Note Currently, the CISCO-ENTITY-FRU-CONTROL-MIB supports only line cards. For additional MIB constraints, see the "CISCO-ENTITY-FRU-CONTROL-MIB" section on page 3-18.

Figure A-6 shows a cefcModuleTable entry for a Gigabit Ethernet line card whose entPhysicalIndex is 24.

Figure A-6 Sample cefcModuleTable Entry

```
cefcModuleEntry.entPhysicalIndex

cefcModuleEntry.24
  cefcModuleAdminStatus = enabled(1)
  cefcModuleOperStatus = ok(2)
  cefcModuleResetReason = manual reset(5)
  cefcModuleStatusLastChangeTime = 7714
```

See the "FRU Status Changes" section on page A-10 for information about how the router generates traps to indicate changes in FRU status.

Generating SNMP Traps

This section provides information about the SNMP traps generated in response to events and conditions on the router, and describes how to identify which hosts are to receive traps.

- Identifying Hosts to Receive Traps
- Configuration Changes
- Environmental Conditions
- FRU Status Changes

Identifying Hosts to Receive Traps

You can use the CLI or SNMP to identify hosts to receive SNMP notifications and to specify the types of notifications they are to receive (traps or informs). For CLI instructions, see the "Enabling Notifications" section on page 4-2. To use SNMP to configure this information, use the following MIB objects:

Use SNMP-NOTIFICATION-MIB objects, including the following, to select target hosts and specify the types of notifications to generate for those hosts:

- snmpNotifyTable—Contains objects to select hosts and notification types:
  - snmpNotifyTag is an arbitrary octet string (a tag value) used to identify the hosts to receive SNMP notifications. Information about target hosts is defined in the snmpTargetAddrTable (SNMP-TARGET-MIB), and each host has one or more tag values associated with it. If a host in snmpTargetAddrTable has a tag value that matches this snmpNotifyTag value, the host is selected to receive the types of notifications specified by snmpNotifyType.
  - snmpNotifyType is the type of SNMP notification to send: trap(1) or inform(2).
- snmpNotifyFilterProfileTable and snmpNotifyFilterTable—Use objects in these tables to create notification filters to limit the types of notifications sent to target hosts.

Use SNMP-TARGET-MIB objects to configure information about the hosts to receive notifications:

- snmpTargetAddrTable—Transport addresses of hosts to receive SNMP notifications. Each entry provides information about a host address, including a list of tag values:
  - snmpTargetAddrTagList—A set of tag values associated with the host address. If a host's tag value matches snmpNotifyTag, the host is selected to receive the types of notifications defined by snmpNotifyType.
- snmpTargetParamsTable—SNMP parameters to use when generating SNMP notifications.

Use the notification enable objects in appropriate MIBs to enable and disable specific SNMP traps. For example, to generate mplsLdpSessionUp or mplsLdpSessionDown traps, the MPLS-LDP-MIB object mplsLdpSessionUpDownTrapEnable must be set to enabled(1).

Configuration Changes

If entity traps are enabled, the router generates an entConfigChange trap (ENTITY-MIB) when the information in any of the following tables changes (which indicates a change to the router configuration):

- entPhysicalTable
- entAliasMappingTable

- entPhysicalContainsTable

Note A management application that tracks configuration changes should occasionally check the value of entLastChangeTime (ENTITY-MIB) to detect any entConfigChange traps that were missed due to throttling or transmission loss.

Enabling Traps for Configuration Changes

To configure the router to generate an entConfigChange trap whenever its configuration changes, enter the following command from the CLI. Use the **no** form of the command to disable the traps.

```
Router(config)# snmp-server enable traps entity
Router(config)# no snmp-server enable traps entity
```

Environmental Conditions

The CISCO-ENVMON-MIB sends the following traps to alert you to conditions detected by environmental sensors in the router:

- 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.

Enabling Environmental Traps

To configure the router to generate traps for environmental conditions, enter the following command from the CLI. Use the **no** form of the command to disable the traps.

```
Router(config)# snmp-server enable traps envmon
Router(config)# no snmp-server enable traps envmon
```

To enable environmental traps through SNMP, set the appropriate notification enable object to true(1). For example, ciscoEnvMonEnableShutdownNotification enables shutdown notifications. Disable the traps by setting the notification object to false(2).

FRU Status Changes

If FRU traps are enabled, the router generates the following traps in response to changes in the status of an FRU. See the CISCO-ENTITY-FRU-CONTROL-MIB for more information about these traps.

- cefcModuleStatusChange—The operational status (cefcModuleOperStatus) of an FRU changes.
- cefcFRUInserted—An FRU is inserted in the chassis. The trap indicates the entPhysicalIndex of the FRU and the container it was inserted in.
- cefcFRURemoved—An FRU is removed from the chassis. The trap indicates the entPhysicalIndex of the FRU and the container it was removed from.

Enabling FRU Traps

To configure the router to generate traps for FRU events, enter the following command from the CLI. Use the **no** form of the command to disable the traps.

```
Router(config)# snmp-server enable traps fru-ctrl
Router(config)# no snmp-server enable traps fru-ctrl
```

To enable FRU traps through SNMP, set cefcMIBEnableStatusNotification to true(1). Disable the traps by setting cefcMIBEnableStatusNotification to false(2).

Using Alarms to Monitor Outages

The Cisco 10000 Series generates alarms to indicate a condition such as the loss of a signal on a SONET path or the activation of a T1 interface on a channelized T3 line card. Alarms also provide status information for router entities, and alert users to conditions that might degrade network performance or cause a failure (an *outage*).

Alarm messages are sent to the console, and information about alarms is also maintained in the CISCO-ENTITY-ALARM-MIB, which is described later in this section. Each physical entity in the entPhysicalTable (ENTITY-MIB) has a set of alarms that define conditions that can occur on the entity.

This section provides basic information about alarms, and it includes the following subsections:

- Viewing Active Alarms Through the CLI
- Using the CISCO-ENTITY-ALARM-MIB to Monitor Alarms

For information about how to monitor router interfaces for problems, see the "Monitoring Router Interfaces" section on page A-16.

Note An alarm indicates a condition, not an event. For example, the alarm "Core critical temperature limit" indicates that the chassis core temperature has reached critical state.

Purpose and Benefits

Using the alarm monitoring feature, you can:

- Monitor when alarms are asserted and cleared
- Obtain alarm history information
- Track alarm statistics and counts
- Generate SNMP traps and syslog messages in response to alarms

MIBs and Alarm Subsystems Used to Monitor Alarms

The router monitors alarms using:

- CISCO-ENTITY-ALARM-MIB—Alarms for physical entities defined in the ENTITY-MIB entPhysicalTable.
- CISCO-SYSLOG-MIB—Contains objects to monitor syslog messages through SNMP.
- Cisco IOS alarm subsystem—Alarms for physical entities.
- Omega alarm subsystem—Alarms for logical entities and interfaces (for example, T1 interfaces on a channelized T3 line card).

Alarm Overview

Each alarm contains the following information:

- Alarm type—A unique code that identifies the alarm.
- Severity—The seriousness of the condition causing the alarm (see the following "Viewing Active Alarms Through the CLI" section for more information).

- Description—Information about the condition that caused the alarm.

Alarm States

An alarm's state indicates the current state of the condition that caused the alarm:

- Asserted—The condition currently exists.
- Cleared—The condition has been resolved.

SNMP uses the following traps to indicate the current state of an alarm:

- ceAlarmAsserted—The condition that caused the alarm still exists.
- ceAlarmCleared—The condition that caused the alarm has been resolved.

By default, an SNMP trap and syslog message are generated each time an alarm is asserted or cleared. You can, however, set the severity level of alarms for which traps and syslog messages are generated, or disable the traps and messages completely (see the "Enabling Traps and Syslog Messages for Alarms" section on page A-16).

Alarm Severity Descriptions

The severity of an alarm indicates the type of condition that the alarm represents:

- critical(1)—A severe, service-affecting condition requiring immediate corrective action.
- major(2)—A hardware or software condition that causes a serious disruption of service, or is on hardware essential to the operation of the router. Although less serious than a critical alarm, a major alarm requires immediate attention to correct the problem.
- minor(3)—A condition or problem that does not affect service, or is on nonessential hardware.
- info(4)—An informational message about an event that improves operation, or an indication of a condition that could cause a problem.

Viewing Active Alarms Through the CLI

To view information about all active alarms on the router, enter the following command from the CLI (where *severity* is the level of alarms to display). The router displays all active alarms with this severity level and higher. If you do not specify *severity*, all active alarms are displayed.

```
Router(config)# show facility-alarm status [ severity ]
```

Using the CISCO-ENTITY-ALARM-MIB to Monitor Alarms

The CISCO-ENTITY-ALARM-MIB allows you to monitor alarms on the router through SNMP.

Note The CISCO-ENTITY-ALARM-MIB monitors alarms only for physical entities defined in the entPhysicalTable (ENTITY-MIB). Alarms for logical entities, such as channelized interfaces, are monitored by Cisco IOS software through syslog.

The MIB contains several tables and objects that provide information about alarms:

- ceAlarmDescrMapTable—Assigns a unique index (ceAlarmDescrIndex) to each physical entity to identify the entity's alarms (see Figure A-7). The table contains an entry for each entity identified by entPhysicalVendorType in the entPhysicalTable (ENTITY-MIB).

- ceAlarmDescrTable—Contains a description of each alarm that every physical entity can generate.
- ceAlarmTable—Lists the alarms currently being asserted by each physical entity on the router, and contains alarm control information for the entity.
- ceAlarmHistTable—Contains a history of the alarms that have occurred on the router.
- ceAlarmCriticalCount—The number of critical alarms currently asserted on the router. The router also maintains numeric counts of major (ceAlarmMajorCount) and minor (ceAlarmMinorCount) alarms.

The MIB also contains objects to control how the router responds to alarms:

- ceAlarmCutOff—Enables you to turn off audible alarms controlled by the SNMP agent. This object functions like an alarm-cutoff switch in the central office. Note that the object has no effect on alarm monitoring and logging, or the generation of SNMP notifications and syslog messages.
- ceAlarmNotifiesEnable—The severity level of alarms that cause an SNMP ceAlarmAsserted or ceAlarmCleared notification to be generated.
- ceAlarmSyslogEnable—The severity level of alarms that cause a syslog message to be generated.

Note See the "Enabling Traps and Syslog Messages for Alarms" section on page A-16 for more information on how to use ceAlarmNotifiesEnable and ceAlarmSyslogEnable.

Interpreting Alarm Information in the CISCO-ENTITY-ALARM-MIB

To obtain information about router alarms from the CISCO-ENTITY-ALARM-MIB, do the following:

Step 1 To determine if any alarms are currently being asserted on the router, read the values of the objects in ceAlarmTable. Each entry in the table contains information about the alarms currently being asserted by each physical entity. Each entry is indexed by the entPhysicalIndex (ENTITY-MIB) of the entity.

Step 2 To obtain information about individual alarms, read the values of the ceAlarmDescrSeverity and ceAlarmDescrText objects. See Figure A-7 for an illustration of how to determine the entity that each alarm is associated with.

Step 3 To determine the total number of alarms currently being asserted (by all entities), read the values of ceAlarmCriticalCount, ceAlarmMajorCount, and ceAlarmMinorCount.

CISCO-ENTITY-ALARM-MIB Examples

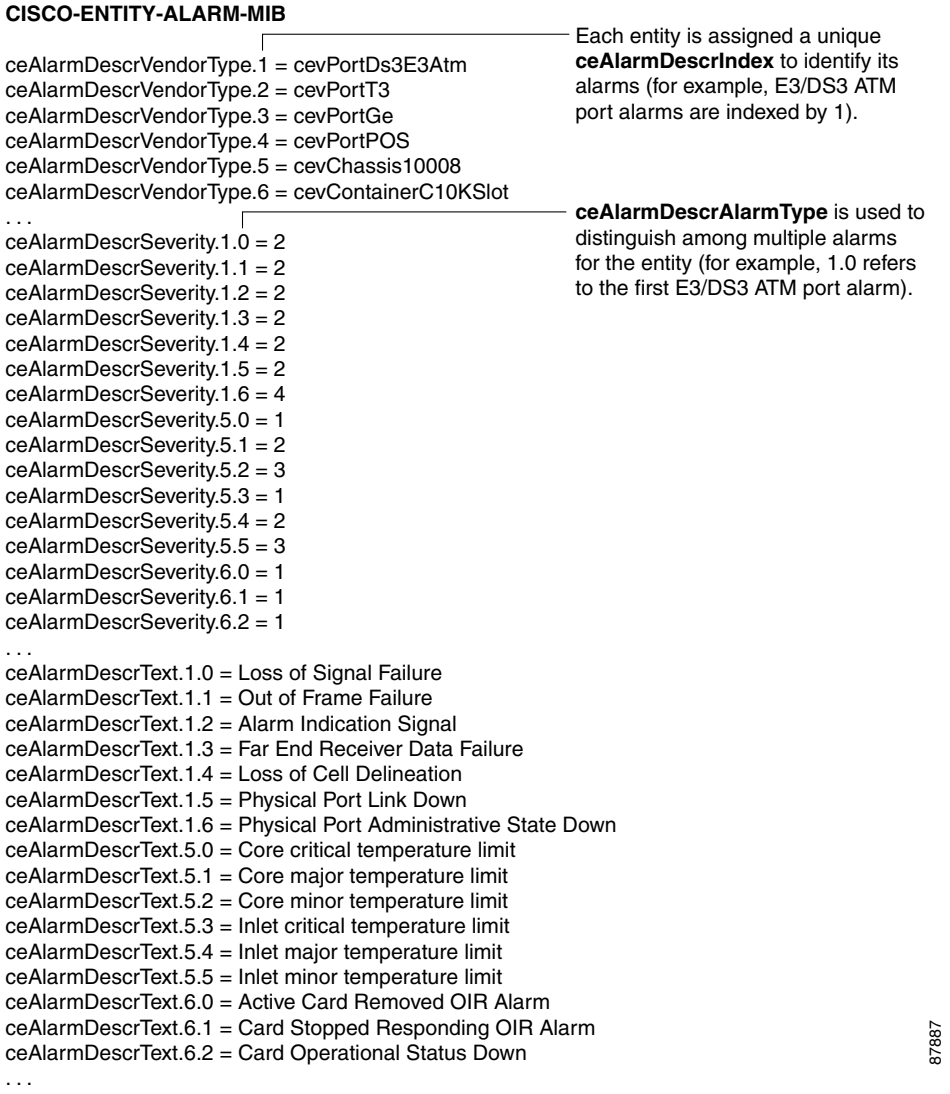
The following is an example of information in the CISCO-ENTITY-ALARM-MIB:

```
ceAlarmDescrVendorType.1 = cevPortDs3E3Atm
ceAlarmDescrVendorType.2 = cevPortT3
ceAlarmDescrVendorType.3 = cevPortGe
ceAlarmDescrVendorType.4 = cevPortPOS
ceAlarmDescrVendorType.5 = cevChassis10008
ceAlarmDescrVendorType.6 = cevContainerC10KSlot
ceAlarmDescrVendorType.7 = cevContainerC10KFanTraySlot
ceAlarmDescrVendorType.8 = cevPowerSupplyC10KAC
ceAlarmDescrVendorType.9 = cevFanTrayC10008
ceAlarmDescrVendorType.10 = cevCpuCreRp
ceAlarmDescrVendorType.11 = cevPortFEIP
ceAlarmDescrVendorType.12 = cevPortOC3SUNI
```

```
ceAlarmDescrVendorType.13 = cevPortOC12SUNI
ceAlarmDescrVendorType.14 = cevPortChOc3Stm1
ceAlarmDescrVendorType.15 = cevPortChOc12
ceAlarmDescrVendorType.16 = cevPortChE1T1
ceAlarmDescrSeverity.1.0 = 2
ceAlarmDescrSeverity.1.1 = 2
ceAlarmDescrSeverity.1.2 = 2
ceAlarmDescrSeverity.1.3 = 2
ceAlarmDescrSeverity.1.4 = 2
ceAlarmDescrSeverity.1.5 = 2
ceAlarmDescrSeverity.1.6 = 4
ceAlarmDescrSeverity.2.0 = 2
ceAlarmDescrSeverity.2.1 = 2
ceAlarmDescrSeverity.2.2 = 2
ceAlarmDescrSeverity.2.3 = 2
ceAlarmDescrSeverity.2.4 = 2
ceAlarmDescrSeverity.2.5 = 2
ceAlarmDescrSeverity.2.6 = 2
ceAlarmDescrSeverity.2.7 = 2
ceAlarmDescrSeverity.2.8 = 2
ceAlarmDescrSeverity.2.9 = 2
ceAlarmDescrSeverity.2.10 = 4
ceAlarmDescrSeverity.3.0 = 1
ceAlarmDescrSeverity.3.1 = 1
. . .
ceAlarmDescrText.1.0 = Loss of Signal Failure
ceAlarmDescrText.1.1 = Out of Frame Failure
ceAlarmDescrText.1.2 = Alarm Indication Signal
ceAlarmDescrText.1.3 = Far End Receiver Data Failure
ceAlarmDescrText.1.4 = Loss of Cell Delineation
ceAlarmDescrText.1.5 = Physical Port Link Down
ceAlarmDescrText.1.6 = Physical Port Administrative State Down
ceAlarmDescrText.2.0 = Far End Remote Alarm Indication Alarm
ceAlarmDescrText.2.1 = Near End Remote Alarm Indication Alarm
ceAlarmDescrText.2.2 = Far End Alarm Indication Signal
ceAlarmDescrText.2.3 = Near End Alarm Indication Signal
ceAlarmDescrText.2.4 = Far End Loss of Frame Failure
ceAlarmDescrText.2.5 = Far End Loss of Signal Failure
ceAlarmDescrText.2.6 = Far End Test Code
ceAlarmDescrText.2.7 = Far End Idle
ceAlarmDescrText.2.8 = Other Failure
ceAlarmDescrText.2.9 = Physical Port Link Down
ceAlarmDescrText.2.10 = Physical Port Administrative State Down
ceAlarmDescrText.3.0 = Physical Port Link Down
ceAlarmDescrText.3.1 = C10K Gigabit Ethernet GBIC missing
. . .
```

Figure A-7 shows how indexes are used to distinguish router alarms.

Figure A-7 CISCO-ENTITY-ALARM-MIB Indexes



The sample CISCO-ENTITY-ALARM-MIB shown above contains the following alarm information (note that only portions of the MIB are shown):

Index	Alarm Text	Severity	Component
1.0	Loss of Signal Failure	2	E3/DS3 ATM port
1.5	Physical Port Link Down	2	E3/DS3 ATM port
5.0	Core critical temperature limit	1	C10008 Chassis
5.5	Inlet minor temperature limit	3	C10008 Chassis
6.0	Active Card Removed OIR Alarm	1	Chassis slot
6.2	Card Operational Status Down	1	Chassis slot

Enabling Traps and Syslog Messages for Alarms

By default, SNMP generates a trap and a syslog message when an alarm is asserted or cleared. You can, however, control the type of alarms for which traps and syslog messages are generated, or disable the traps and messages completely, by performing the instructions in the following sections.

Enabling Traps for Alarms

To enable or disable SNMP traps for alarms, do either of the following. By default, SNMP generates a trap when an alarm is asserted or cleared.

- At the CLI, enter the following command. Use the **no** form of the command to disable the traps.

```
Router(config)# snmp-server enable traps alarms
Router(config)# no snmp-server enable traps alarms
```

- Through SNMP, set the following CISCO-ENTITY-ALARM-MIB object:

  ceAlarmNotifiesEnable—The severity level of alarms that cause an SNMP trap to be generated: critical(1), major(2), minor(3), or info(4). Disable traps by setting ceAlarmNotifiesEnable to 0.

  For example, set ceAlarmNotifiesEnable to major to generate a trap for major and critical alarms; or, set ceAlarmNotifiesEnable to minor to generate a trap for minor, major, and critical alarms. See the "Viewing Active Alarms Through the CLI" section on page A-12 for descriptions of alarm severities.

Note The CISCO-ENTITY-ALARM-MIB monitors alarms only for physical entities defined in the ENTITY-MIB entPhysicalTable. Alarms for logical entities, such as channelized interfaces, are monitored by Cisco IOS software through syslog.

Enabling Syslog Messages for Alarms

By default, the router logs a syslog message each time an alarm is asserted or cleared. You can, however, use the following CISCO-SYSLOG-MIB object to configure the types of alarms for which a syslog message is generated:

- ceAlarmSyslogEnable—The severity level of alarms to generate a syslog message for: critical(1), major(2), minor(3), or info(4). See the "Viewing Active Alarms Through the CLI" section on page A-12 for descriptions of alarm severities. To disable these syslog messages, set ceAlarmSyslogEnable to 0.

In addition, the following CISCO-SYSLOG-MIB objects provide SNMP notification when syslog messages are logged in response to alarms:

- clogNotificationsEnabled—Specifies whether to generate a notification when a syslog message is logged. Set this object to true(1) to enable the notifications, or false(2) to disable them.
- clogMessageGenerated—SNMP notification sent when a syslog message is generated.

Monitoring Router Interfaces

This section provides information about how to monitor the status of router interfaces to see if there is a problem or a condition that might affect service on the interface. To determine if an interface is Down or experiencing problems, you can:

Check the Interface's Operational and Administrative Status

To check the status of an interface, view the following IF-MIB objects for the interface:

- ifAdminStatus—The administratively configured (desired) state of an interface. Use ifAdminStatus to enable or disable the interface.
- ifOperStatus—The current operational state of an interface.

Monitor linkDown and linkUp Traps

To determine if an interface has failed, you can monitor linkDown and linkUp traps for the interface. See the "Enabling Interface linkUp/linkDown Traps" section on page A-17 for instructions on how to enable these traps.

- linkDown—Indicates that an interface has failed or is about to fail.
- linkUp—Indicates that an interface is no longer in the Down state.

Check the Interface for Alarms

You can also check to see if an interface is currently asserting either of the following alarms, which indicate that the interface is down and can not send or receive traffic:

- Physical Port Link Down—Indicates that the operational state of an interface is Down.
  - For Ethernet, OC-*x*, OC-*x* ATM, OC-*x* POS, and STM-*x* line cards, the alarm has a severity level of critical(1).
  - For line cards that support serial lines (for example, E1/T1, T3, and E3/DS3 ATM), the alarm has a severity level of major(2) or minor(3).
- Physical Port Administrative State Down—Indicates that the administrative (desired) state of an interface is Down. When the administrative state is Down, the operational state also changes to Down.
  - For all line cards, this alarm is severity level info(4).

For information about how to monitor alarms, see the "Using Alarms to Monitor Outages" section on page A-11. Also see the "Using the CISCO-ENTITY-ALARM-MIB to Monitor Alarms" section and the "Interpreting Alarm Information in the CISCO-ENTITY-ALARM-MIB" section on page A-13.

Note SNMP generates a ceAlarmAsserted or ceAlarmCleared trap when an alarm is generated or cleared. You can read the CISCO-ENTITY-ALARM-MIB to see if any interfaces are asserting alarms.

Enabling Interface linkUp/linkDown Traps

To configure SNMP to send a notification when a router interface changes state to Up (ready) or Down (not ready), perform the following steps to enable linkUp and linkDown traps:

Step 1 Issue the following CLI command to enable linkUp and linkDown traps for most, but not necessarily all, interfaces:

```
Router(config)# snmp-server enable traps snmp linkdown linkup
```

Step 2 View the setting of the ifLinkUpDownTrapEnable object (IF-MIB ifXTable) for each interface to determine if linkUp and linkDown traps are enabled or disabled for that interface.

Step 3 To enable linkUp and linkDown traps on an interface, set ifLinkUpDownTrapEnable to enabled(1). For information about how to configure the router to send linkDown traps only for the lowest layer of an interface, see the "SNMP Trap Filtering for linkDown Traps" section on page A-18.

Note Some interface layers do not support linkDown traps (for example, some ATM layers).

Step 4 To enable the Internet Engineering Task Force (IETF) standard for linkUp and linkDown traps, issue the following command. (The IETF standard is based on RFC 2233.)

```
Router(config)# snmp-server trap link ietf
```

Step 5 To enable linkUp and linkDown traps on ATM subinterfaces, issue the following command:

```
Router(config)# snmp-server enable traps atm subif
```

Step 6 To enable linkUp and linkDown traps on an ATM permanent virtual circuit (PVC), issue the following commands. In the first command, **interval** specifies the minimum interval between successive traps, and **fail-interval** specifies the minimum interval for storing failed time stamps.

```
Router(config)# snmp-server enable traps atm pvc interval seconds fail-interval seconds
Router(config)# interface atm slot/subslot/port
Router(config-if)# pvc vpi/vci
Router(config-if-atm-vc)# oam-pvc manage
```

Step 7 To disable traps, use the **no** form of the appropriate command.

SNMP Trap Filtering for linkDown Traps

Use the SNMP trap filtering feature to filter linkDown traps so that SNMP sends a linkDown trap only if the main interface goes down. If an interfaces goes down, all of its subinterfaces go down, which results in numerous linkDown traps for each subinterface. This feature filters out those subinterface traps.

This feature is turned off by default. To enable the SNMP trap filtering feature, issue the following CLI command. Use the **no** form of the command to disable the feature.

```
[no] snmp ifmib trap throttle
```

Monitoring PXF Utilization

This section describes how to use SNMP to monitor parallel express forwarding network processor (PXF) utilization on the router by:

- Determining PXF Utilization and Efficiency
- Monitoring PXF Performance Thresholds and Restarts

Purpose and Benefits

The CISCO-ENTITY-PFE-MIB provides SNMP access to performance information for the packet forwarding engine (PFE), which accelerates certain IP features in order to improve network performance. The MIB contains objects to monitor PFE utilization and efficiency. On the Cisco 10000 Series, the PFE is the parallel express forwarding network processor (PXF), which is part of the performance routing engine (PRE).

The MIB provides the following benefits:

- Summarizes the PXF utilization and efficiency information in the following CLI command:

  ```
  show hardware pxf cpu context
  ```

- Provides information about performance trends for 1-minute, 5-minute, and 15-minute intervals
- Maintains a 24-hour history of PXF utilization and efficiency
- Measures PXF performance against user-configurable utilization and efficiency thresholds and generates an event when a threshold is exceeded or the PXF is restarted

MIBs Used to Monitor PXF Utilization

- CISCO-ENTITY-PFE-MIB

Determining PXF Utilization and Efficiency

CISCO-ENTITY-PFE-MIB utilization and efficiency objects are used to measure PXF performance:

- PXF *utilization* is the percentage of the PXF currently being used for processing. As PXF processing increases, utilization rises from 0 to 100 percent.
- PXF *efficiency* measures how well the PXF is performing. The higher the value, the greater the PXF's efficiency. During normal operating conditions, PXF efficiency is typically 100 percent. As efficiency degrades, this value decreases.

To determine PXF utilization and efficiency on the router, view the information in the following MIB tables:

- cePfePerfCurrentTable—Utilization and efficiency percentages: current, 1-minute, and 5-minute.
- cePfePerfIntervalTable—Performance statistics for the past 24 hours, in 15-minute intervals. The table holds 96 measurement intervals, less if the PXF has not been running for 24 hours.

  The start of each 15-minute interval is based on the time that the PXF was last started or restarted, which may not match the start of a quarter-hour increment in real time (for example, 10:45 or 11:15). For example, if the PXF was started at 10:20, subsequent 15-minute intervals start at 10:35, 10:50, and so on.
- cePfePerfTotalTable—Utilization and efficiency for the past 24 hours.

Monitoring PXF Performance Thresholds and Restarts

You can use the CISCO-ENTITY-PFE-MIB to set thresholds for PXF utilization and efficiency, and monitor PXF performance against those thresholds. SNMP compares PXF performance (ceCpfePerfCurrentTable) to the thresholds (cePfePerfConfigTable) and generates an event when PXF utilization or efficiency reaches or exceeds a threshold. You can log the event to an event history table (cePfeHistTable), generate an SNMP notification, do both, or take no action. A PXF event is also generated each time the PXF is restarted.

For example, if the PXF 1-minute utilization measurement (cePfePerfCurrent1MinUtilization) reaches or exceeds its threshold (cePfePerfThld1MinUtilization), SNMP generates a thld1MinUtilizationEvent. See the MIB object HistEventType for descriptions of events.

Perform the following steps to track PXF utilization and efficiency by setting and monitoring PXF thresholds:

Step 1 Use cePfePerfConfigTable to define acceptable thresholds for PXF utilization and efficiency.

Step 2 Set cePfeHistNotifiesEnable to one of the following values to specify what SNMP should do when a threshold is exceeded or the PXF is restarted:

- none(1)—SNMP takes no action. This is the default.
- log(2)—Create an entry in the cePfeHistTable.
- notify(3)—Send an SNMP notification.
- logAndNotify(4)—Create a cePfeHistTable entry and send an SNMP notification.

Step 3 To log events to cePfeHistTable, use cePfeHistTableSize to specify the maximum number of entries to allow in the table. When the table becomes full, each new event overwrites the oldest entry in the table.

Step 4 If you set cePfeHistNotifiesEnable to log(2) or logAndNotify(4), you can view the cePfeHistTable for information about exceeded thresholds and PXF restarts.

Preprovisioning Line Cards

This section provides information about the process of preprovisioning line cards and its affect on SNMP data. The preprovisioning feature preconfigures a line card slot for a particular type of line card before the line card is actually inserted into the chassis. The system adds a basic configuration for the line card to the system's running configuration file, and then applies this configuration to the line card when it is actually inserted in the chassis.

Note You cannot use SNMP to preprovision a line card slot. You must use the CLI **card** command to do that. For instructions, see the *Cisco 10000 Series Router Line Card Slot Preprovisioning* feature description under the Cisco 10000 Series Router New Features documentation link on CCO.

Installing a Preprovisioned Line Card

When a line card is inserted in the Cisco 10000 chassis, the following occurs:

- If the inserted line card matches the type of line card preprovisioned for the slot, the system applies the preprovisioned configuration to the line card.
- If the line card slot was not preprovisioned, the system applies a basic configuration to the line card and adds that configuration to the running configuration file.
- If the line card slot was preprovisioned for one type of line card, but another type of line card was inserted, the system replaces the preprovisioned configuration (in the running configuration file) with a basic configuration for the line card that was actually inserted.

To find out the type of card that a slot is configured to use, enter the **show running-config** command.

Affected MIBs

The information in the following MIB tables is affected when a preprovisioned line card is inserted into the Cisco 10000 chassis, or you enter CLI commands that affect line card operation:

- ENTITY-MIB (entPhysicalTable and entAliasMappingTable)
- CISCO-ENTITY-ASSET-MIB (ceAssetTable)
- CISCO-ENTITY-FRU-CONTROL-MIB (cefcModuleTable)

Replacing Line Cards—MIB State Characteristics

When you replace a line card in the Cisco 10000 chassis, the contents of MIBs are affected as follows. If you replace a line card with:

- Another type of line card (for example, if you replace a Gigabit Ethernet line card with a channelized T3 line card), the original line card information is removed from the MIBs.
- Another line card of the *same* type, the MIBs retain the original line card configuration. This enables you to replace a line card without losing its configuration. For example, if you replace a Gigabit Ethernet line card that has 50 subinterfaces, the MIBs retain the information about the line card and its subinterfaces. To replace the original line card information in the MIBs, see the following steps.

To replace a line card with another line card of the same type and remove the original line card information from the MIBs, perform the following steps:

Step 1 To shut down the line card, issue the following CLI command (where *slot_number* is 1 through 8):

```
hw-module slot slot_number shutdown
```

Step 2 Wait 30 seconds for the line card failure LED to light.

Step 3 Remove the line card from the chassis.

Note If you do not plan to install another line card in the slot for some period of time, we recommend that you issue the **no card** command (see Step 4).

Step 4 To remove line card information from the router configuration and MIBs, issue the following CLI command. For example, the command **no card 5/0** removes configuration information for the line card in slot 5.

```
no card slot/subslot
```

Step 5 (Optional) To verify that the line card configuration information was removed from the MIBs, view the contents of the MIBs.

Step 6 Insert a new line card in the chassis slot.

Step 7 To activate the newly installed line card, issue the following CLI command:

```
no hw-module slot slot_number shutdown
```

Performing Bulk-File Retrieval

This section describes how to use SNMP to perform bulk retrieval of large amounts of data from the router. You can use this feature to transfer information between the SNMP agent and manager (such as QoS statistics, interface statistics, and entPhysicalTable entries).

Purpose and Benefits

In previous releases, a management application was required to enter lengthy sequences of SNMP **get-next** or **get-bulk** requests to retrieve large amounts of data from the router.

The SNMP bulk-file retrieval feature simplifies this process, and may result in better performance. To perform a bulk-file retrieval:

1. Define the characteristics of the bulk file.
2. Specify the data to include in the bulk file.
3. Create the bulk file and fill it with the data you want to transfer.
4. Use the File Transfer Protocol (FTP) utility to copy the bulk file from the router to another system.

To perform the bulk-file retrieval, either:

- Enter SNMP **set** and **get** requests from the host (see the "SNMP Commands" section on page A-23 for command examples).
- Create a management application that issues SNMP **set** and **get** requests.

The MIB enhancements feature also includes a Java applet that retrieves the router's ifTable using the bulk-retrieval process (see the "Java Applet" section on page A-25).

MIBs Used for Bulk-File Retrieval

- CISCO-BULK-FILE-MIB
- CISCO-FTP-CLIENT-MIB

Bulk-File Retrieval Processing Steps

This section describes how to perform a bulk-file retrieval. For examples of this process, see the "SNMP Commands" section on page A-23 and the "Java Applet" section on page A-25. For detailed information about MIB objects, their characteristics, and valid values, see the MIB.

Step 1 Define the characteristics of a bulk file by creating a row in the CISCO-BULK-FILE-MIB cbfDefineFileTable:

a. Determine a unique index to assign to the row.

b. Set cbfDefineFileTable objects:

cbfDefineFileEntryStatus = createAndGo(4) or createAndWait(5)
cbfDefineFileName = *bulk_file_name*
cbfDefineFileStorage = ephemeral(1)
cbfDefineFileFormat = bulkASCII(3)

Step 2 Define the data to include in the bulk file by creating a row in cbfDefineObjectTable:

a. Determine a unique index to assign to the row.

b. Set cbfDefineObjectTable objects:

cbfDefineFileObjectStatus = createAndGo(4) or createAndWait(5)
cbfDefineObjectID = *the OID of the object instance or table column*
cbfDefineObjectClass = object(1) or lexicaltable(2)

c. (Optional) To include specific table rows in the bulk file, set the following MIB objects. To use this option, you must set cbfDefineObjectClass = lexicaltable(2).

cbfDefineObjectTableInstance = *starting table row*
cbfDefineObjectNumEntries = *number of table rows to include*

For example, to include rows 2 through 12 of ifTable, set cbfDefineObjectTableInstance = ifTable.2 and cbfDefineObjectNumEntries = 10.

Step 3 Create the bulk file, fill it with data, and check its status:

a. Set cbfDefineFileNow = create(3).

b. Perform a **get-next** on cbfStatusFileState, using the bulk file's cbfDefineFileIndex.

c. Do not take any action until cbfStatusFileState = ready(2).

Step 4 Configure FTP to copy the bulk file to an FTP server by creating a row in the CISCO-FTP-CLIENT-MIB cfcRequestTable:

a. Determine a unique index to use for the row.

b. Set cfcRequestTable objects:

cbfRequestEntryStatus = createAndWait(5) or createAndGo(4)
cfcRequestOperation = putASCII(2)
cfcRequestLocalFile = *name of the bulk file on the router*
cfcRequestRemoteFile = *name (and path) to copy bulk file to on destination FTP server*
cfcRequestServer = *IP address or fully qualified name of FTP server*
cfcRequestUser = *a valid user name for FTP server*
cfcRequestPassword = *password for FTP user name*

c. Set cfcRequestEntryStatus to active(1) to activate the row, which starts the bulk-file transfer.

d. Check cfcRequestResult to view the result of the FTP operation.

SNMP Commands

Figure A-1 shows sample SNMP commands for performing bulk-file retrieval. These commands are samples only; your commands will be different. Check the MIB for valid values. The numbers in the figure correspond to the steps in the "Bulk-File Retrieval Processing Steps" section on page A-22. Notes and background information appear in Table A-1.

Note This example makes use of the SNMP EMANATE tool (from SNMP Research International). In the following commands, *rtr_IP_addr* is the IP address of the router.

Figure A-1 Sample SNMP Command for Bulk-File Retrieval

```
1  setany -v2c rtr_IP_addr private cbfDefineFileEntryStatus.1 -i 4
   setany -v2c rtr_IP_addr private cbfDefineFileName.1 -D "QoSstats"
   setany -v2c rtr_IP_addr private cbfDefineFileStorage.1 -i 1
   setany -v2c rtr_IP_addr private cbfDefineFileFormat.1 -i 3

2  setany -v2c rtr_IP_addr private cbfDefineObjectEntryStatus.1.3 -i 4
   setany -v2c rtr_IP_addr private cbfDefineObjectID.1.3 -d 1.3.6.1.2.1.4.20
   setany -v2c rtr_IP_addr private cbfDefineObjectClass.1.3 -i 2

3  setany -v2c rtr_IP_addr private cbfDefineFileNow.1 -i 3
   getone -v2c rtr_IP_addr private cbfStatusFileState.1.1

4  setany -v2c rtr_IP_addr private cfcRequestEntryStatus.1 -i 5
   setany -v2c rtr_IP_addr private cfcRequestOperation.1 -i 2
   setany -v2c rtr_IP_addr private cfcRequestLocalFile.1 -D "QoSstats"
   setany -v2c rtr_IP_addr private cfcRequestRemoteFile.1 -D "C10kQoS"
   setany -v2c rtr_IP_addr private cfcRequestServer.1 -D "stats.cisco.com"
   setany -v2c rtr_IP_addr private cfcRequestUser.1 -D "JoeSmith"
   setany -v2c rtr_IP_addr private cfcRequestPassword.1 -D "bluefi$h"
   setany -v2c rtr_IP_addr private cfcRequestEntryStatus.1 -i 1
```

69731

Step 1 A row is created for a bulk file named QoSstats and the row is placed in the Active state.

An index of 1 is assigned to the row and used to access table objects for the row (for example, cbfDefineFileEntryStatus**.1**, cbfDefineFileName**.1**, cbfDefineFileStorage**.1**, and so on).

The bulk file stores data only until it is read, and its format is human-readable ASCII.

Step 2 A row is created and its index is **.1.3** (cbfDefine**File**Index and cbfDefine**Object**Index).

The MIB settings specify that data in the ipAddrTable is to be included in the bulk file.

Step 3 The bulk file is created, and cbfDefineFileIndex is used to check that the bulk file was created.

Step 4 The commands set up an FTP **put** request to copy the bulk file (QoSstats) to the stats.cisco.com server to a file named C10kQoS. By default, the file is copied to the specified user's home directory; however, you can specify another directory. For example, cfcRequestRemoteFile = /C10Kstats/QoSstats copies the bulk file to the directory /C10Kstats.

If you use SNMP commands, consider the following:

- For each row in a table, you must determine a unique index to use to access table objects for the row. The index must be unique among all rows in the table. You must define the index when you create the row.
- To create a row in a table:
  - Append the row's index to the table's *xxxEntryStatus* object
  - Set *xxxEntryStatus* = createAndGo(4) or createAndWait(5)

  The system creates the row and assigns the specified index to the row. For example, the following command creates a row in cbfDefineFileTable and sets the row to the Inactive state. The row is assigned an index of 1.

  ```
  setany -v2c rtr_IP_addr private cbfDefineFileEntryStatus.1 -i 5
  ```

  Use this index value to access the row's other MIB objects (for example, cbfDefineFileName.1, cbfDefineFileStorage.1, and so on).

Java Applet

To use the Java applet to perform a bulk-file retrieval of the ifTable, follow these steps:

1. Make sure the router is connected to a workstation that supports the Java 2 platform (which is required to run the applet).
2. Go to the Cisco FTP site at the following URL:

   ftp://ftp-eng.cisco.com/auto/ftp/omega/cheops
3. Copy the following file from the FTP site to your workstation:

   10kApplets.jar
4. At the workstation, enter the following command to launch the applet. (Note that applet.BulkFileRetrieval tells the system to run the bulk-file retrieval class or program within the JAR file.)

   ```
   java -cp <JAR_file_location> applet.BulkFileRetrieval
   ```
5. Enter the following information in the window that is displayed:
   - IP address of the Ethernet port on the router
   - SNMP version and community
   - IP address of the destination FTP server to transfer the bulk file to
   - A valid username and password for the server
   - Home directory of the specified user (the bulk file is copied here)
6. Click **Retrieve BULK-FILE** to start the bulk-file retrieval.

   The system copies the ifTable to a bulk file named ifTable–bulkFile<*MonthDay–HourMinSec*> (for example, ifTable–bulkFileJan3–17hr9min16sec), then copies the bulk file to the home directory of the specified user on the destination FTP server.
7. Click the **BULK-FILE Data** tab to view the bulk file.

Figure A-2 shows the Java applet for performing a bulk-file retrieval. Numbers correspond to steps in the "Bulk-File Retrieval Processing Steps" section on page A-22. See Table 0-1 for background information about certain steps in the applet.

Figure A-2 Java Applet for Bulk-File Retrieval

```
public int createCbfDefineFileRow(String bulkFileName) {

(1a) int[] cbfDefineFileTableIndex = {getRandomNumber()};
     String indexValue = snmp.makeOIDFromArray(cbfDefineFileTableIndex);

(1b) snmp.snmpSet_addVarbind("cbfDefineFileEntryStatus",
                             indexValue,
                             RowStatusEnum_createAndGo);

     snmp.snmpSet_addVarbind("cbfDefineFileName",
                             indexValue,
                             bulkFileName);

     snmp.snmpSet_addVarbind("cbfDefineFileStorage",
                             indexValue,
                             FileStorageEnum_ephemeral);

     snmp.snmpSet_addVarbind("cbfDefineFileFormat",
                             indexValue,
                             FileFormatEnum_bulkASCII);

     snmp.snmpSet_go();
}


public boolean createCbfDefineObjectRow(int bulkFileId,
                                        String objectClass,
                                        String objectId) {

(2a) int[] cbfDefineObjectTableIndex = {bulkFileId, getRandomNumber()};
     String indexValue = snmp.makeOIDFromArray(cbfDefineObjectTableIndex);

(2b) snmp.snmpSet_addVarbind("cbfDefineObjectEntryStatus",
                             indexValue,
                             RowStatusEnum_createAndGo);

     snmp.snmpSet_addVarbind("cbfDefineObjectClass", indexValue, objectClass);

     snmp.snmpSet_addVarbind("cbfDefineObjectID", indexValue, objectId);

     snmp.snmpSet_go();
     }


     public boolean startAndMonitorBulkFileCreation(int bulkFileId) {

(3a)      String indexValue = "." + bulkFileId;
          snmp.snmpSet_addVarbind("cbfDefineFileNow",
                                  indexValue,
                                  FileNowEnum_create);
          snmp.snmpSet_go();

(3b)      SnmpVarBind result = snmp.snmpGetNextObject("cbfStatusFileState", indexValue);
          String fileStateIndex = snmp.extractIndexValues(result);
          int fileStateValue = result.getValue()
```

69732

Figure A-3 Java Applet for Bulk-File Retrieval (continued)

```
3c  while (fileStateValue == FileStateEnum_running) {
              sleep(FileStatePollInterval);
              result = snmp.snmpGetObject("cbfStatusFileState", fileStateIndex);
              fileStateValue = result.getValue();
         }

         boolean returnResult = determineResult(fileStateValue);
         return returnResult;
    }


    public boolean startAndMonitorBulkFileTransfer(String bulkFileName,
         String     bulkFileType,
         String     ftpServerAddr,
         String     ftpServerUsername,
         String     ftpServerPassword) {

4a       int[] cfcRequestTableIndex = {getRandomNumber()};
         String indexValue = snmp.makeOIDFromArray(cfcRequestTableIndex);

4b       snmp.snmpSet_addVarbind("cfcRequestEntryStatus",
                                 indexValue,
                                 RowStatusEnum_createAndWait);

         snmp.snmpSet_addVarbind("cfcRequestOperation",
                                 indexValue,
                                 bulkFileType);

         snmp.snmpSet_addVarbind("cfcRequestLocalFile",
                                 indexValue,
                                 bulkFileName);

         snmp.snmpSet_addVarbind("cfcRequestRemoteFile",
                                 indexValue,
                                 bulkFileName);

         snmp.snmpSet_addVarbind("cfcRequestServer",
                                 indexValue,
                                 ftpServerAddr);

         snmp.snmpSet_addVarbind("cfcRequestUser",
                                 indexValue,
                                 bulkFileUsername);

         snmp.snmpSet_addVarbind("cfcRequestPassword",
                                 indexValue,
                                 ftpServerPassword);

         snmp. snmpSet_go();
```

69733

Figure A-4 Java Applet for Bulk-File Retrieval (continued)

```
4c  snmp.snmpSet_addVarbind("cfcRequestEntryStatus",
                                indexValue,
                                RowStatusEnum_active);

        returnStatus = snmp.snmpSet_go();

4d      SnmpVarBind result = snmp.snmpGetObject("cfcRequestResult", indexValue);
        int requestResultState = result.getValue();
        int timeToCompletion = 0;
        while (requestResultState == cfcRequestResult_pending ||
                       timeToCompletion < BulkFileTransferTimeout) {
                  sleep(cfcRequestStatePollInterval);
                  timeToCompletion+=cfcRequestStatePollInterval;
                  result = snmp.snmpGetObject("cfcRequestResult", indexValue);
                  requestResultState = result.getValue();
        }

        boolean returnResult = determineResult(fileStateValue);
        return returnResult;
    }
```

69734

Step 1a	Generate a random number to use as the index.
Step 1b	MIB objects are set as follows: cbfDefineFileStorage = ephemeral(1) cbfDefineFileFormat = bulkASCII(3)
Step 2a	Generate a random number to use as the index.
Step 3b	Use cbfDefineFileIndex to access cbfStatusFileState.
Step 4c	Poll cbfStatusFileState until the state changes from running(1) to ready(2).
Step 5d	Poll cfcRequestResult until the bulk-file transfer is no longer pending.

Monitoring Quality of Service

This section provides an example of how to use SNMP to access QoS configuration information and statistics on the router. It contains the following sections:

- Configuring QoS, page A-29
- Accessing QoS Configuration Information and Statistics, page A-29
- Monitoring QoS, page A-34
- Sample QoS Applications, page A-41

Purpose and Benefits

Previously, the only way to access QoS configuration information and statistics was to enter **show** commands at the CLI.

With the enhanced management feature, you can use SNMP to access QoS configuration information and statistics on the router. This means that you can now collect and store QoS information for use in management applications. You can also use bulk-file transfer to copy the information to another system.

MIBs Used for QoS

- CISCO-CLASS-BASED-QOS-MIB

Configuring QoS

You configure QoS through the command line interface (CLI). For instructions, see the *Cisco 10000 Series Router Software Configuration Guide*, "Configuring Quality of Service."

Accessing QoS Configuration Information and Statistics

The CISCO-CLASS-BASED-QOS-MIB provides access to QoS configuration information and statistics. Although you cannot use SNMP to configure QoS on the router, you can use SNMP to access QoS configuration information that has been configured through the CLI.

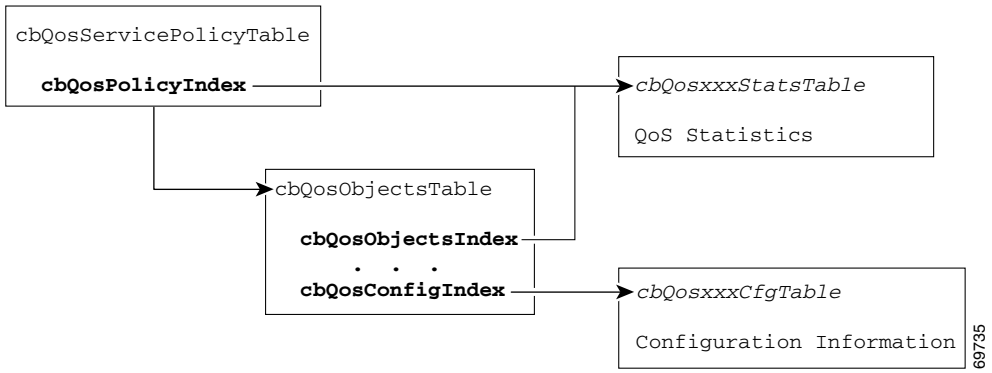
QoS Indexes

The indexes for accessing QoS configuration information and QoS statistics are:

- cbQosPolicyIndex—System-assigned index that identifies a policy map attached to an interface. (When attached to an interface, a policy map is known as a *service policy*.)
- cbQosObjectsIndex—System-assigned index that identifies each unique run-time instance of a QoS feature (for example, policy map, class map, match statement, and feature action).
- cbQosConfigIndex—System-assigned index that identifies each unique configuration of a QoS feature (for example, a class map or police action). Note that QoS objects with the same configuration share the same cbQosConfigIndex.
- cbQosREDValue—The IP precedence or IP differentiated services code point (DSCP) of a Weighted Random Early Detection (WRED) action. It is used as the index for configuration information and statistics for each RED class.

Figure A-5 shows how these indexes provide access to QoS configuration information and statistics.

Figure A-5 Cisco 10000 Series Router QoS Indexes



To access QoS configuration information and statistics for a particular QoS feature:

1. Look in cbQosServicePolicyTable and find the cbQosPolicyIndex assigned to the policy in which the feature is used.
2. Use cbQosPolicyIndex to access the cbQosObjectsTable, and find the cbQosObjectsIndex and cbQosConfigIndex assigned to the QoS feature.
   - Use cbQosConfigIndex to access configuration tables (cbQos*xxx*CfgTable) for information about the feature.
   - Use cbQosPolicyIndex and cbQosObjectsIndex to access QoS statistics tables (cbQos*xxx*StatsTable) for information about the QoS feature.

Sample QoS Configuration Settings

This section contains figures that show how QoS configuration settings are stored in CISCO-CLASS-BASED-QOS-MIB tables:

- Figure A-6 shows the sample QoS configuration that the other figures are based on.
- Figure A-7 on page A-32 shows the service policy and objects table.
- Figure A-8 on page A-33 shows policy map, class map, and police action configuration information.
- Figure A-9 on page A-34 shows RED class configuration settings for an ATM interface.

The figures in this section show information grouped by QoS object; however, the actual output of an SNMP query might show QoS information similar to the following. This is only a partial display of all QoS information for the configuration in Figure A-6.

```
c10k# getmany -v3 10.86.0.94 test-user ciscoCBQosMIB

cbQosIfType.1047 = subInterface(2)
cbQosIfType.1052 = subInterface(2)
cbQosPolicyDirection.1047 = input(1)
cbQosPolicyDirection.1052 = output(2)
cbQosIfIndex.1047 = 36
cbQosIfIndex.1052 = 36
cbQosFrDLCI.1047 = 0
cbQosFrDLCI.1052 = 0
cbQosAtmVPI.1047 = 0
cbQosAtmVPI.1052 = 0
cbQosAtmVCI.1047 = 0
cbQosAtmVCI.1052 = 0
cbQosConfigIndex.1047.1047 = 1045
cbQosConfigIndex.1047.1048 = 1025
cbQosConfigIndex.1047.1050 = 1027
cbQosConfigIndex.1047.1051 = 1046
cbQosConfigIndex.1052.1052 = 1045
cbQosConfigIndex.1052.1053 = 1025
cbQosConfigIndex.1052.1055 = 1027
cbQosConfigIndex.1052.1056 = 1046
cbQosObjectsType.1047.1047 = policymap(1)
cbQosObjectsType.1047.1048 = classmap(2)
cbQosObjectsType.1047.1050 = matchStatement(3)
cbQosObjectsType.1047.1051 = police(7)
cbQosObjectsType.1052.1052 = policymap(1)
cbQosObjectsType.1052.1053 = classmap(2)
cbQosObjectsType.1052.1055 = matchStatement(3)
cbQosObjectsType.1052.1056 = police(7)
cbQosParentObjectsIndex.1047.1047 = 0
cbQosParentObjectsIndex.1047.1048 = 1047
cbQosParentObjectsIndex.1047.1050 = 1048
```

```
    cbQosParentObjectsIndex.1047.1051 = 1048
    cbQosParentObjectsIndex.1052.1052 = 0
    cbQosParentObjectsIndex.1052.1053 = 1052
    cbQosParentObjectsIndex.1052.1055 = 1053
    cbQosParentObjectsIndex.1052.1056 = 1053
    cbQosPolicyMapName.1045 = pm-1Meg
    cbQosPolicyMapDesc.1045 =
    cbQosCMName.1025 = class-default
    cbQosCMDesc.1025 =
    cbQosCMInfo.1025 = matchAny(3)
         . . .
```

Figure A-6 GigabitEthernet QoS Configuration—CLI show Commands

```
c10k# show class-map
 Class Map match-any class-default (id 0)
   Match any

 Class Map match-any cm1 (id2)
  Description: class map #1
   Match ip  dscp 48

c10k# show policy-map
  Policy Map pm1
    Class cm1
      shape 1200000
      random-detect dscp-based
      random-detect dscp 32 202 8000 20
      random-detect dscp 44 200 6000 22
      random-detect dscp 61 201 7000 22
  Policy Map pm-1Meg
    Class class-default
      police 1000000 8000 8000 conform-action transmit exceed-action drop

c10k# show policy-map interface
 GigabitEthernet1/0/0.1

  Service-policy input: pm-1Meg (1057)

    Class-map: class-default (match-any) (1058/0)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any  (1060)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Police:
        1000000 bps, 8000 limit, 8000 extended limit
        conformed 0 packets, 0 bytes; action: transmit
        exceeded 0 packets, 0 bytes; action: drop

  Service-policy output: pm-1Meg (1062)

    Class-map: class-default (match-any) (1063/0)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any  (1065)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Output queue: 0/8192; 0/0 packets/bytes output, 0 drops
      Police:
        1000000 bps, 8000 limit, 8000 extended limit
        conformed 0 packets,0 bytes; action: transmit
        exceeded 0 packets, 0 bytes; action: drop
```

69736

Figure A-7 GigabitEthernet QoS—Service Policy and Objects Tables

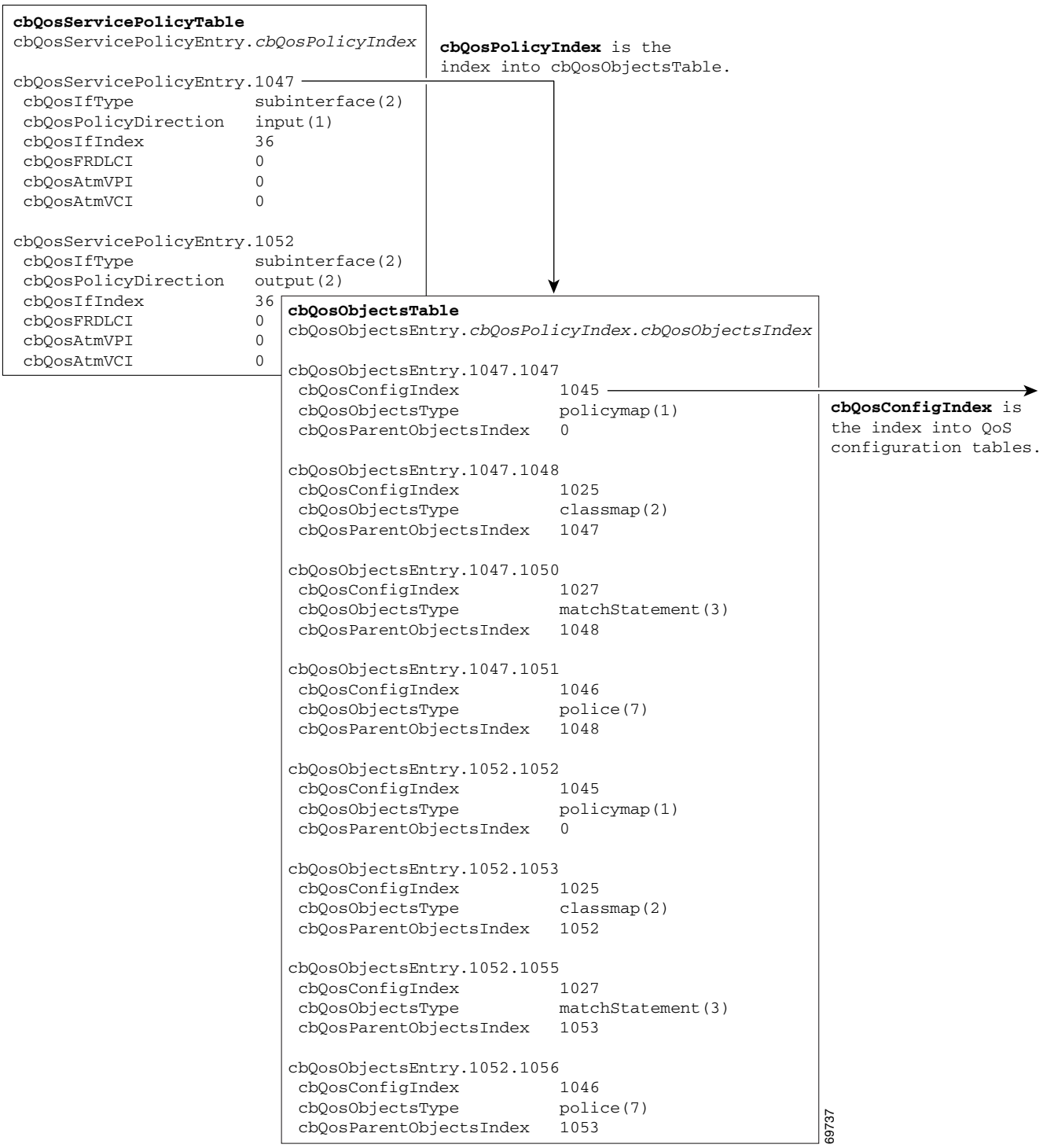


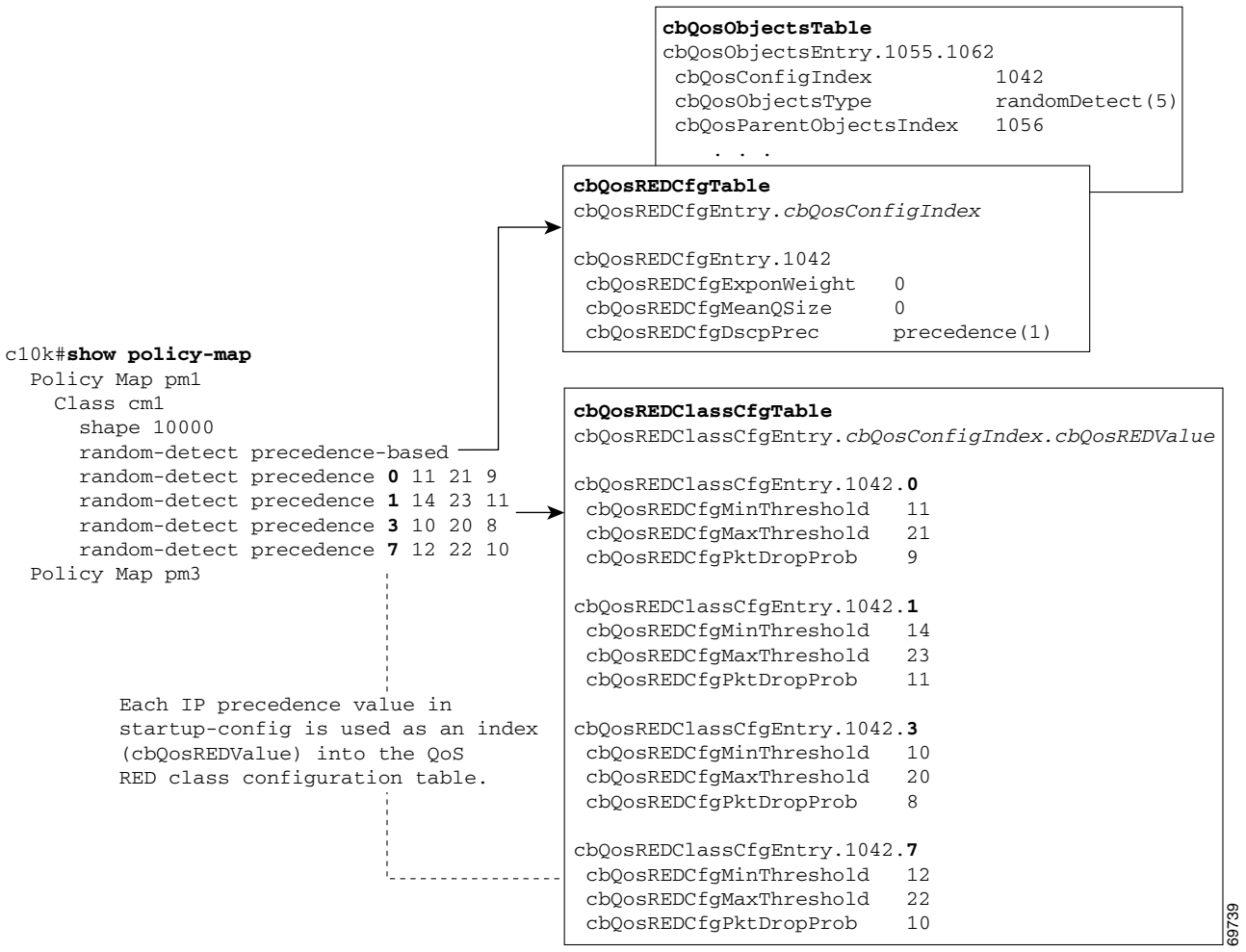
Figure A-8 GigabitEthernet QoS—Policy Map, Class Map, and Police Action Configuration Objects

Note the following about the sample QoS configuration:

- Because the policy map pm-1Meg is attached to an input and an output interface:
  - Two cbQosObjectsIndex values are used (one for the input interface, the other for the output)
  - A single cbQosConfigIndex is used for both the input and the output objects
- Policy maps that are not attached to an interface are not included with SNMP data or displayed by the **show policy-map interface** command. This is why pm-1Meg is shown but pm1 is not.
- The default class map is always included with the SNMP data.
- Class maps that have no action defined are not included with the SNMP data. This is why cm1 is not part of cbQosCMCfgTable.

Figure A-9 shows an example of RED configuration information stored in MIB tables. Note that this configuration is applied to an ATM interface, not Gigabit Ethernet.

Figure A-9 ATM QoS—RED Configuration Objects



Monitoring QoS

This section provides information about how to monitor QoS on the router by checking the QoS statistics in the MIB tables described in Table A-1. For information about how to determine the amount of traffic to bill customers for, see the "Billing Customers for Traffic" section on page A-44.

Note The CISCO-CLASS-BASED-QOS-MIB might contain more information than what is displayed in the output of CLI **show** commands.

Table A-1 QoS Statistics Tables

QoS Table	Statistics
cbQosCMStatsTable	Class Map—Counts of packets, bytes, and bit rate before and after QoS policies are executed. Counts of dropped packets and bytes.
cbQosMatchStmtStatsTable	Match Statement—Counts of packets, bytes, and bit rate before executing QoS policies.
cbQosPoliceStatsTable	Police Action—Counts of packets, bytes, and bit rate that conforms to, exceeds, and violates police actions.
cbQosQueueingStatsTable	Queueing—Counts of discarded packets and bytes, and queue depths.
cbQosTSStatsTable	Traffic Shaping—Counts of delayed and dropped packets and bytes, the state of a feature, and queue size.
cbQosREDClassStatsTable	Random Early Detection—Counts of packets and bytes dropped when queues were full, and counts of bytes and octets transmitted.

Considerations for Processing QoS Statistics

The router maintains 64-bit counters for most QoS statistics. However, some QoS counters are implemented as a 32-bit counter with a 1-bit overflow flag. In the following figures, these counters are shown as 33-bit counters.

When accessing QoS statistics in counters, consider the following:

- SNMPv2c or SNMPv3 applications—Access the entire 64 bits of the QoS counter through *cbQosxxx64* MIB objects.
- SNMPv1 applications—Access QoS statistics in the MIB as follows:
  - Access the lower 32 bits of the counter through *cbQosxxx* MIB objects.
  - Access the upper 32 bits of the counter through *cbQosxxxOverflow* MIB objects.

QoS Statistics Tables

The figures in this section show the counters in CISCO-CLASS-BASED-QOS-MIB statistics tables:

- Figure A-10 shows the counters in the cbQosCMStatsTable and the indexes for accessing these and other statistics.
- Figure A-11 shows the counters in cbQosMatchStmtStatsTable, cbQosPoliceStatsTable, cbQosQueueingStatsTable, cbQosTSStatsTable, and cbQosREDClassStatsTable.

See the "Sample QoS Statistics" section on page A-37 for examples of QoS statistics stored in tables.

For ease-of-use, the following figures show some counters as a single object even though the counter is implemented as three objects. For example, `cbQosCMPrePolicyByte` is implemented as:

```
cbQosCMPrePolicyByteOverflow
cbQosCMPrePolicyByte
cbQosCMPrePolicyByte64
```

Note Due to implementation features, some of the QoS statistics counters might wrap before they reach the maximum value they can accommodate.

Figure A-10 QoS Class Map Statistics and Indexes

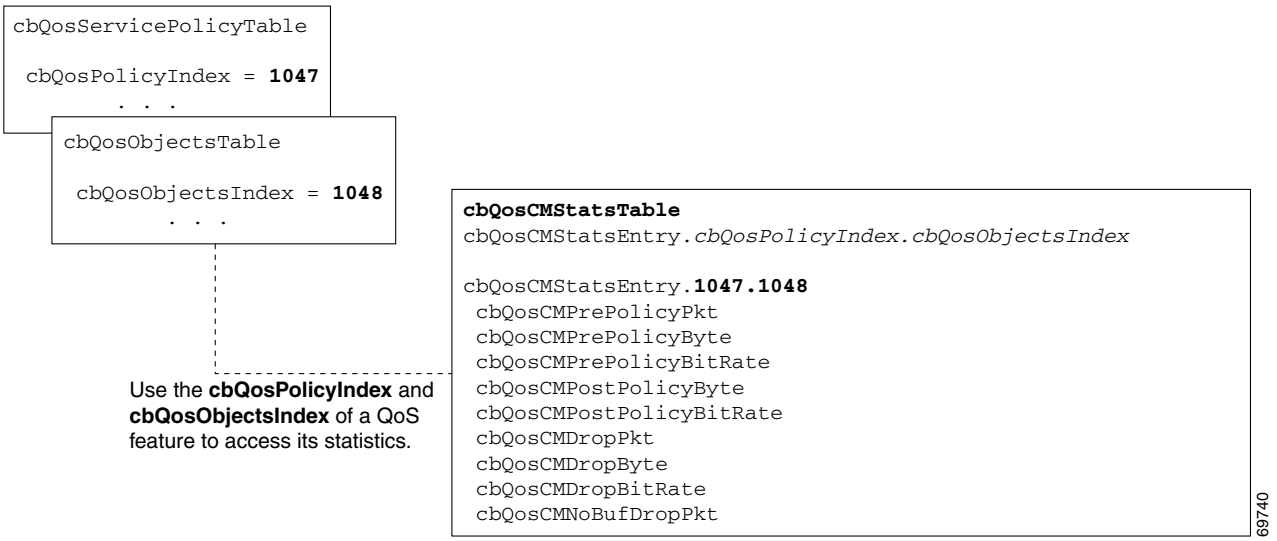
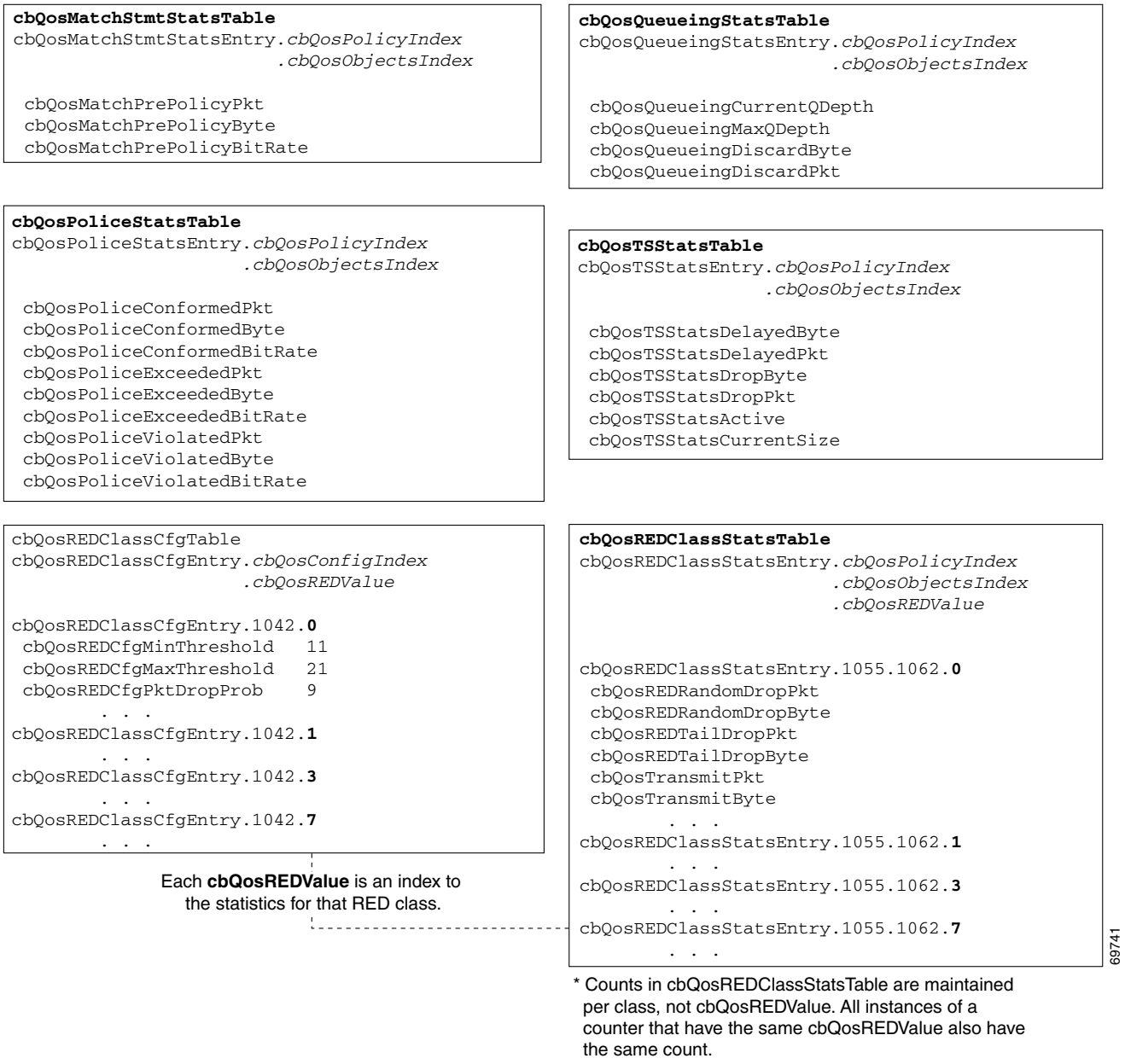


Figure A-11 QoS Statistics Tables



* Counts in cbQosREDClassStatsTable are maintained per class, not cbQosREDValue. All instances of a counter that have the same cbQosREDValue also have the same count.

Sample QoS Statistics

This section contains figures that show how QoS statistics are displayed in show commands and stored in CISCO-CLASS-BASED-QOS-MIB tables.

- Figure A-12 shows an example of QoS statistics displayed by the **show policy-map interface** command.
- Figure A-13 on page A-39 shows class map statistics for the input service policy.
- Figure A-14 on page A-40 shows class map statistics for the output service policy.

- Figure A-15 on page A-41 shows match statement statistics for the input and output service policies.

Figure A-12 Sample QoS Statistics—CLI show Commands

```
c10k# show running-config interface GigabitEthernet 1/0/0.1
Building configuration...

Current configuration : 188 bytes
!
interface GigabitEthernet1/0/0.1
 encapsulation dot1Q 1
 ip address 10.1.0.2 255.255.255.0
 no ip directed-broadcast
 service-policy input pm-1meg
 service-policy output pm-1meg
end

c10k# show policy-map interface
 GigabitEthernet1/0/0.1

  Service-policy input: pm-1meg (2428)

    Class-map: class-default (match-any) (2429/0)
      4801508 packets, 667409423 bytes
      5 minute offered rate 1668000 bps, drop rate 667000 bps
      Match: any  (2431)
        4801508 packets, 667409423 bytes
        5 minute rate 1668000 bps
      Police:
        1000000 bps, 8000 limit, 8000 extended limit
        conformed 2878916 packets, 400169135 bytes; action: transmit
        exceeded 1922592 packets, 267240288 bytes; action: drop

  Service-policy output: pm-1meg (2433)

    Class-map: class-default (match-any) (2434/0)
      14259374 packets, 1925015267 bytes
      5 minute offered rate 1639000 bps, drop rate 640000 bps
      Match: any  (2436)
        14259374 packets, 1925015267 bytes
        5 minute rate 1639000 bps
      Output queue: 0/8192; 3698585/514006021 packets/bytes output, 0 drops
      Police:
        1000000 bps, 8000 limit, 8000 extended limit
        conformed 3517209 packets, 474822992 bytes; action: transmit
         exceeded 10742165 packets, 1450192275 bytes; action: drop c10k#
```

69742

Figure A-13 QoS Class Map Statistics—Input Service Policy

```
c10k# show policy-map interface
 GigabitEthernet1/0/0.1

  Service-policy input: pm-1meg (2428)

    Class-map: class-default (match-any) (2429/0)
      4801508 packets, 667409423 bytes
      5 minute offered rate 1668000 bps, drop rate 667000 bps
      Match: any  (2431)
        4801508 packets, 667409423 bytes
        5 minute rate 1668000 bps
      Police:
        1000000 bps, 8000 limit, 8000 extended limit
        conformed 2878916 packets, 400169135 bytes; action: transmit
        exceeded 1922592 packets, 267240288 bytes; action: drop
```

```
cbQosCMCfgTable
 cbQosCMName = class-default
 cbQosCMInfo = matchAny(3)

cbQosObjectsTable
 cbQosObjectsIndex = 2004
 cbQosObjectsType = classmap(2)
 cbQosParentObjectsIndex = 2003

cbQosServicePolicyTable
 cbQosPolicyIndex = 2003
 cbQosIfType = subinterface(2)
 cbQosPolicyDirection = input(1)
```

cbQosCMStatsTable
cbQosCMStatsEntry.*cbQosPolicyIndex*.*cbQosObjectsIndex*

cbQosCMStatsEntry.**2003.2004**	
cbQosCMPrePolicyPktOverflow	0
cbQosCMPrePolicyPkt	4801508
cbQosCMPrePolicyPkt64	0x0004943e4
cbQosCMPrePolicyByteOverflow	0
cbQosCMPrePolicyByte	667409423
cbQosCMPrePolicyByte64	0x027c7dc0f
cbQosCMPrePolicyBitRate	1668000
cbQosCMPostPolicyByteOverflow	0
cbQosCMPostPolicyByte	401004108
cbQosCMPostPolicyByte64	0x017e6d64c
cbQosCMPostPolicyBitRate	1001000
cbQosCMDropPktOverflow	0
cbQosCMDropPkt	1922592
cbQosCMDropPkt64	0x0001d5620
cbQosCMDropByteOverflow	0
cbQosCMDropByte	266405315
cbQosCMDropByte64	0x00fe105c3
cbQosCMDropBitRate	667000
cbQosCMNoBufDropPktOverflow	0
cbQosCMNoBufDropPkt	0
cbQosCMNoBufDropPkt64	0x000000000

69743

Figure A-14 QoS Class Map Statistics—Output Service Policy

```
c10k# show policy-map interface
 GigabitEthernet1/0/0.1
        . . .

Service-policy output: pm-1meg (2433)

   Class-map: class-default (match-any) (2434/0)
       14259374 packets, 1925015267 bytes
       5 minute offered rate 1639000 bps, drop rate 640000 bps
     Match: any  (2436)
       14259374 packets, 1925015267 bytes
       5 minute rate 1639000 bps
     Output queue: 0/8192; 3698585/514006021 packets/bytes output, 0 drops
     Police:
       1000000 bps, 8000 limit, 8000 extended limit
       conformed 3517209 packets, 474822992 bytes; action: transmit
       exceeded 10742165 packets, 1450192275 bytes; action: drop
```

```
cbQosCMCfgTable
 cbQosCMName = class-default
 cbQosCMInfo = matchAny(3)
```

```
cbQosObjectsTable
 cbQosObjectsIndex = 1909
 cbQosObjectsType = classmap(2)
 cbQosParentObjectsIndex = 1908
```

```
cbQosServicePolicyTable
 cbQosPolicyIndex = 1908
 cbQosIfType = subinterface(2)
 cbQosPolicyDirection = output(2)
```

cbQosCMStatsTable
cbQosCMStatsEntry.*cbQosPolicyIndex*.*cbQosObjectsIndex*

cbQosCMStatsEntry.**1908.1909**	
cbQosCMPrePolicyPktOverflow	0
cbQosCMPrePolicyPkt	14259374
cbQosCMPrePolicyPkt64	0x000d994ae
cbQosCMPrePolicyByteOverflow	0
cbQosCMPrePolicyByte	1925015267
cbQosCMPrePolicyByte64	0x072bd66e3
cbQosCMPrePolicyBitRate	1639000
cbQosCMPostPolicyByteOverflow	0
cbQosCMPostPolicyByte	475598027
cbQosCMPostPolicyByte64	0x01c590ccb
cbQosCMPostPolicyBitRate	999000
cbQosCMDropPktOverflow	0
cbQosCMDropPkt	10742165
cbQosCMDropPkt64	0x000a3e995
cbQosCMDropByteOverflow	0
cbQosCMDropByte	1449417240
cbQosCMDropByte64	0x056645a18
cbQosCMDropBitRate	640000
cbQosCMNoBufDropPktOverflow	0
cbQosCMNoBufDropPkt	0
cbQosCMNoBufDropPkt64	0x000000000

69744

Figure A-15 QoS Match Statement Statistics

Sample QoS Applications

This section presents examples of sample code showing how to retrieve information from the CISCO-CLASS-BASED-QOS-MIB to use for QoS billing operations. You can use these examples to help you develop billing applications. The sample code shows how to:

- Checking Customer Interfaces for Service Policies
- Retrieving QoS Billing Information

Checking Customer Interfaces for Service Policies

This section describes a sample algorithm that checks the CISCO-CLASS-BASED-QOS-MIB for customer interfaces with service policies, and marks those interfaces for further application processing (such as billing for QoS services).

The algorithm uses two SNMP **get-next** requests for each customer interface. For example, if the router has 2000 customer interfaces, 4000 SNMP **get-next** requests are required to determine whether those interfaces have transmit and receive service policies associated with them.

Note This algorithm is for informational purposes only. Your application needs may be different.

Check the MIB to see which interfaces are associated with a customer. Create a pair of flags to show whether a service policy has been associated with the transmit and receive directions of a customer interface. Mark non-customer interfaces TRUE (so no more processing is required for them).

```
FOR each ifEntry DO
  IF (ifEntry represents a customer interface) THEN
     servicePolicyAssociated[ifIndex].transmit = FALSE;
     servicePolicyAssociated[ifIndex].receive = FALSE;
  ELSE
     servicePolicyAssociated[ifIndex].transmit = TRUE;
     servicePolicyAssociated[ifIndex].receive = TRUE;
  END-IF
END-FOR
```

Examine the cbQosServicePolicyTable and mark each customer interface that has a service policy attached to it. Also note the direction of the interface.

```
x = 0;
done = FALSE;
WHILE (!done)
  status = snmp-getnext (
           ifIndex = cbQosIfIndex.x,
           direction = cbQosPolicyDirection.x
  );
  IF (status != 'noError') THEN
     done = TRUE
  ELSE
     x = extract cbQosPolicyIndex from response;
     IF (direction == 'output') THEN
       servicePolicyAssociated[ifIndex].transmit = TRUE;
     ELSE
       servicePolicyAssociated[ifIndex].receive = TRUE;
     END-IF
  END-IF
END-WHILE
```

Manage cases in which a customer interface does not have a service policy attached to it.

```
FOR each ifEntry DO
  IF (!servicePolicyAssociated[ifIndex].transmit) THEN
     Perform processing for customer interface without a transmit service policy.
  END-IF
  IF (!servicePolicyAssociated[ifIndex].receive) THEN
     Perform processing for customer interface without a receive service policy.
  END-IF
END-FOR
```

Retrieving QoS Billing Information

This section describes a sample algorithm that uses the CISCO-CLASS-BASED-QOS-MIB for QoS billing operations. The algorithm periodically retrieves post-policy input and output statistics, combines them, and sends the result to a billing database.

The algorithm uses the following:

- One SNMP **get** request per customer interface—to retrieve the ifAlias.
- Two SNMP **get-next** requests per customer interface—to retrieve service policy indexes.
- Two SNMP **get-next** requests per customer interface for each object in the policy—to retrieve post-policy bytes. For example, if there are 100 interfaces and 10 objects in the policy, the algorithm requires 2000 **get-next** requests (2 x 100 x 10).

Note This algorithm is for informational purposes only. Your application needs may be different.

Set up customer billing information.

```
FOR each ifEntry DO
  IF (ifEntry represents a customer interface) THEN
     status = snmp-getnext (id = ifAlias.ifIndex);
     IF (status != 'noError') THEN
         Perform error processing.
     ELSE
        billing[ifIndex].isCustomerInterface = TRUE;
        billing[ifIndex].customerID = id;
        billing[ifIndex].transmit   = 0;
        billing[ifIndex].receive    = 0;
     END-IF
  ELSE
     billing[ifIndex].isCustomerInterface = FALSE;
  END-IF
END-FOR
```

Retrieve billing information.

```
x = 0;
done = FALSE;
WHILE (!done)
  response = snmp-getnext (
             ifIndex = cbQosIfIndex.x,
             direction = cbQosPolicyDirection.x
  );
  IF (response.status != 'noError') THEN
     done = TRUE
  ELSE
     x = extract cbQosPolicyIndex from response;
     IF (direction == 'output') THEN
        billing[ifIndex].transmit = GetPostPolicyBytes (x);
     ELSE
        billing[ifIndex].receive = GetPostPolicyBytes (x);
     END-IF
  END-IF
END-WHILE
```

Determine the number of post-policy bytes for billing purposes.

```
GetPostPolicyBytes (policy)
  x = policy;
  y = 0;
  total = 0;
  WHILE (x == policy)
     response = snmp-getnext (type = cbQosObjectsType.x.y);
     IF (response.status == 'noError')
        x = extract cbQosPolicyIndex from response;
        y = extract cbQosObjectsIndex from response;
        IF (x == policy AND type == 'classmap')
           status = snmp-get (bytes = cbQosCMPostPolicyByte64.x.y);
           IF (status == 'noError')
```

```
                    total += bytes;
                END-IF
            END-IF
        END-IF
    END-WHILE
RETURN total;
```

Billing Customers for Traffic

This section describes how to use SNMP QoS information to determine the amount of traffic to bill to your customers. It also includes a scenario for demonstrating that a QoS service policy attached to an interface is policing traffic on that interface.

This section describes the following topics:

- Determining the Amount of Traffic to Bill to a Customer, page A-44
- Scenario for Demonstrating QoS Traffic Policing, page A-45

Input and Output Interface Counts

The router maintains information about the number of packets and bytes that are received on an input interface and transmitted on an output interface. When a QoS service policy is attached to an interface, the router applies the rules of the policy to traffic on the interface and increments the packet and bytes counts on the interface.

The following CISCO-CLASS-BASED-QOS-MIB objects provide interface counts:

- cbQosCMDropPkt and cbQosCMDropByte (cbQosCMStatsTable)—Total number of packets and bytes that were dropped because they exceeded the limits set by the service policy. These counts include only those packets and bytes that were dropped because they exceeded service policy limits. The counts do not include packets and bytes dropped for other reasons.
- cbQosPoliceConformedPkt and cbQosPoliceConformedByte (cbQosPoliceStatsTable)—Total number of packets and bytes that conformed to the limits of the service policy and were transmitted.

Determining the Amount of Traffic to Bill to a Customer

Perform these steps to determine how much traffic on an interface is billable to a particular customer:

Step 1 Determine which service policy on the interface applies to the customer.

Step 2 Determine the index values of the service policy and class map used to define the customer's traffic. You will need this information in the following steps.

Step 3 Access the cbQosPoliceConformedPkt object (cbQosPoliceStatsTable) for the customer to determine how much traffic on the interface is billable to this customer.

Step 4 (Optional) Access the cbQosCMDropPkt object (cbQosCMStatsTable) for the customer to determine how much of the customer's traffic was dropped because it exceeded service policy limits.

Scenario for Demonstrating QoS Traffic Policing

This section describes a scenario that demonstrates the use of SNMP QoS statistics to determine how much traffic on an interface is billable to a particular customer. It also shows how packet counts are affected when a service policy is applied to traffic on the interface.

To create the scenario, follow these steps, each of which is described in the sections that follow:

1. Create and attach a service policy to an interface.
2. View packet counts before the service policy is applied to traffic on the interface.
3. Issue a **ping** command to generate traffic on the interface. Note that the service policy is applied to the traffic.
4. View packet counts after the service policy has been applied to determine how much traffic to bill the customer for:
   - Conformed packets—The number of packets within the range set by the service policy and for which you can charge the customer.
   - Exceeded or dropped packets—The number of packets that were not transmitted because they were outside the range of the service policy. These packets are not billable to the customer.

Note In the above scenario, the Cisco 10000 Series is used as an interim device (that is, traffic originates elsewhere and is destined for another device).

Service Policy Configuration

This scenario uses the following policy-map configuration. For information on how to create a policy map, see "Configuring Quality of Service" in the *Cisco 10000 Series Router Software Configuration Guide*.

```
policy-map police-out
  class BGPclass
    police 8000 1000 2000 conform-action transmit exceed-action drop

interface GigabitEthernet1/0/0.10
 description VLAN voor klant
 encapsulation dot1Q 10
 ip address 10.0.0.17 255.255.255.248
 service-policy output police-out
```

Packet Counts before the Service Policy Is Applied

The following CLI and SNMP output shows the interface's output traffic before the service policy is applied:

CLI Command Output

```
c10k# show policy-map interface g6/0/0.10

GigabitEthernet6/0/0.10

Service-policy output: police-out

   Class-map: BGPclass (match-all)
      0 packets, 0 bytes
      30 second offered rate 0 bps, drop rate 0 bps
```

```
      Match: access-group 101
      Police:
        8000 bps, 1000 limit, 2000 extended limit
        conformed 0 packets, 0 bytes; action: transmit
        exceeded 0 packets, 0 bytes; action: drop

   Class-map: class-default (match-any)
      4 packets, 292 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: any
      Output queue: 0/8192; 2/128 packets/bytes output, 0 drops
```

SNMP Output

```
c10k# getone -v2c 10.86.0.63 public ifDescr.65
ifDescr.65 = GigabitEthernet6/0/0.10-802.1Q vLAN subif
```

Generating Traffic

The following set of **ping** commands generates traffic:

```
c10k# ping
Protocol [ip]:
Target IP address: 10.0.0.18
Repeat count [5]: 99
Datagram size [100]: 1400
Timeout in seconds [2]: 1
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.

Sending 100, 1400-byte ICMP Echos to 10.0.0.18, timeout is 1 seconds:
.!!.!..!.!..!.!..!.!..!.!..!.!.!..!.!..!.!..!.!..!.!..!.!..!.!.!..
!.!..!.!..!.!.!..!.!..!.!..!.!
Success rate is 42 percent (42/100), round-trip min/avg/max = 1/1/1 ms
```

Packet Counts after the Service Policy Is Applied

After you generate traffic using the **ping** command, look at the number of packets that exceeded and conformed to the committed access rate (CAR) set by the **police** command:

- 42 packets conformed to the police rate and were transmitted
- 57 packets exceeded the police rate and were dropped

The following CLI and SNMP output show the counts on the interface after the service policy is applied. (In the output, conformed and exceeded packet counts are shown in boldface.)

CLI Command Output

```
c10k# show policy-map interface g6/0/0.10

GigabitEthernet6/0/0.10

Service-policy output: police-out

   Class-map: BGPclass (match-all)
      198 packets, 281556 bytes
      30 second offered rate 31000 bps, drop rate 11000 bps
      Match: access-group 101
      Police:
        8000 bps, 1000 limit, 2000 extended limit
```

```
            conformed 42 packets, 59892 bytes; action: transmit
            exceeded 57 packets, 81282 bytes; action: drop

         Class-map: class-default (match-any)
            15 packets, 1086 bytes
            30 second offered rate 0 bps, drop rate 0 bps
            Match: any
            Output queue: 0/8192; 48/59940 packets/bytes output, 0 drops
```

SNMP Output

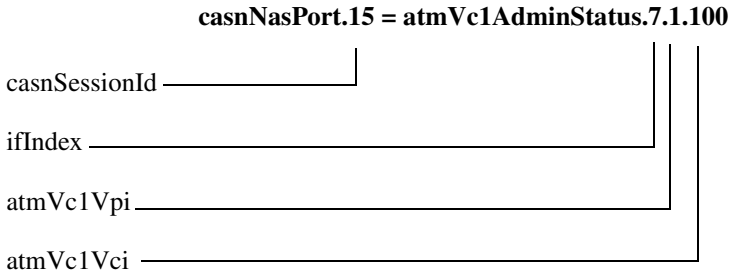
```
c10k# getmany -v2c 10.86.0.63 public ciscoCBQosMIB
        . . .
        cbQosCMDropPkt.1143.1145 = 57
        . . .
        cbQosPoliceConformedPkt.1143.1151 = 42
        . . .
```

Using CISCO-AAA-SESSION-MIB

The following object support was added to the CISCO-AAA-SESSION-MIB to improve interface mapping sessions:

- casnNasPort—Identifies a particular conceptual row associated with the session identified by casnSessionId. The conceptual row that this object points to represents a port that is used to transport a session. If the port transporting the session cannot be determined, the value of this object will be zeroDotZero

  For example, a session is established using an ATM PVC. If the ifIndex of the ATM interface is 7 and the VPI/VCI values of the PVC are 1, 100 respectively, then the value of this object is (in this example):



  Where atmVc1AdminStatus is the first accessible object of the atmVcTable of the ATM-MIB.

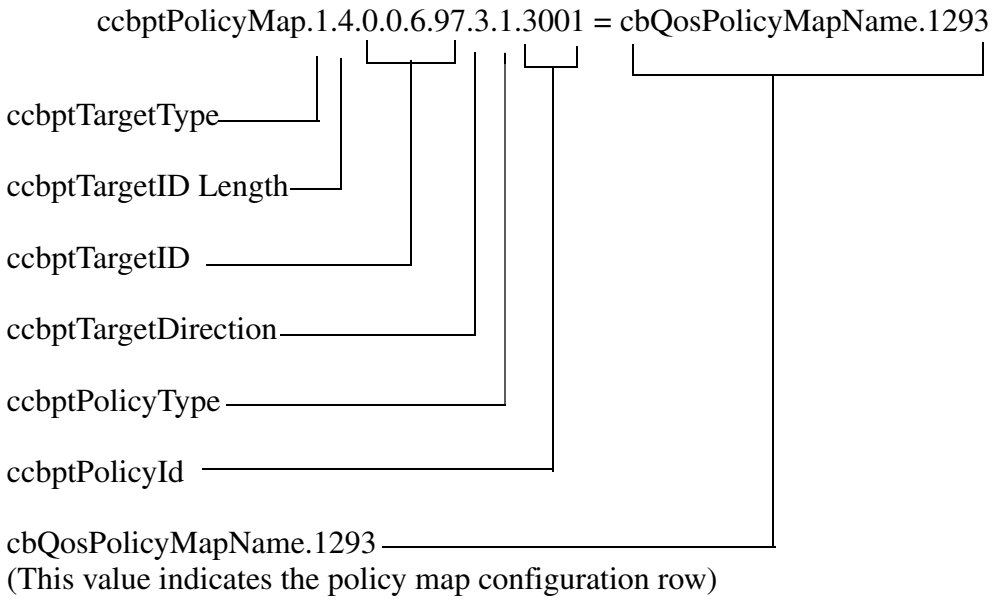
- casnVaiIfIndex—Identifies the ifIndex of the Virtual Access Interface (VAI) that is associated with the PPP session. This interface may not be represented in the IF-MIB in which case the value of this object will be zero.

Using CISCO-CBP-TARGET-MIB

The CISCO-CBP-TARGET-MIB contains objects that define textual conventions for representing targets which have class based policy mappings. A target can be any logical interface or entity to which a class based policy is able to be applied.

The ccbptTarget is a series of octets that should be interpreted according to the value of ccbptTargetType.

The following is only one example of an index with the type genIf(1) and how to decode index values corresponding to config mapping data output.



The figure above indicates the mapping of the index portion of the object identifier (OID) for an instance of the ccbptPolicyMap object. Each portion of the index is defined below.

```
Config Policy Mapping Data
------------------------------
    ccbptPolicyMap.1.4.0.0.6.97.3.1.3001 = cbQosPolicyMapName.1293
```

Where from left to right:

- ccbptTargetType—Value of **1** indicates the ccbptTargetType which is genIf(1). The target type indicates that the value contained in the ccbptTargetId is an ifIndex value.
- ccbptTargetId Length—Value of **4** indicates that the length of the ccbptTargetId to follow is 4 bytes. The ccbptTargetId is defined in the MIB as a variable length OCTET-STRING representing it in the index of a table requires that it be preceded by the length of an octet string.
- ccbptTargetId—Value of 0.0.6.97 indicates the target ID. The length of the third index is determined by the value in the second byte of the entire index (in this example, the length of the target ID is 4 bytes). For supported ccbptTargetID values, see Possible Values for ccbptTargetID.

Numerical Value for the ifIndex Example

The numerical value of this if Index ccbptTargetID, 0.0.6.97, is defined below.

(bits) 0—— 7 8—— 15 16—— 23 24—— 31

97 6 0 0

97 +
6* 2^8 (256) = 6*256 = 1536

1633 = numeric value of the ccbptTargetID, 0.0.6.97

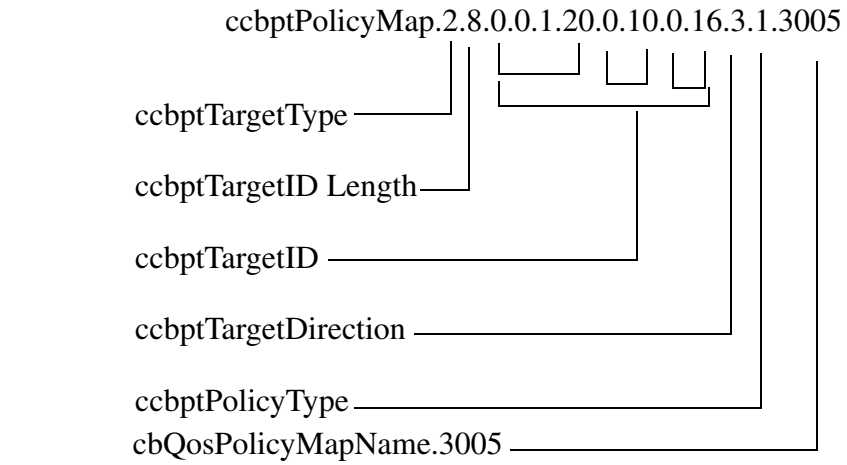
- ccbptTargetDirection—Value of **3** indicates the ccbptTarget output direction.
- ccbptPolicyType—Value of **1** indicates the ccbptPolicyType which is ciscoCbQos(1).
- ccbptPolicyId—Value of **3001** indicates the ccbptPolicyId which is the policy index integer for the policy instance applied to the target.

  Value is an unsigned32 (1.. 4294967295) which in this example the ccbptPolicyId equals the cbQosPolicyIndex which is the index to the CbQosService PolicyTable from the CISCO-CLASS-BASED-QOS-MIB.
- **cbQosPolicyMapName.1293** value indicates the row in the cbQosPolicyMapTable describing the configuration of the policy map applied to the output direction of this ccbptTargetId.

Possible Values for ccbptTargetID

The supported ccbptTargetID values are:

- For genIf(1), OCTECSTRING (SIZE(4)) – ifIndex (4d). Where the (4d) value is a four-byte decimal for the length of the ccbptTargetId in our example.
- For atmPvc(2), OCTET STRING (SIZE(8)) – ATM PVC (4d:2d:2d). Where the ATM PVC has a ccbptTargetId length of 8 bytes (4d:2d:2d). For example:



4d: = 0.0.1.20 = ifIndex
2d:= 0.10 = VPI
2d: = 0.16 = VCI

  - For frDlci(3), OCTET STRING(SIZE(6)) – Frame Relay ifIndex is first 4 bytes and DLCI is the last 2 bytes (4d:2d)
  - For controlPlane(4), OCTET STRING(SIZE(4)) – Control Plane Entity (4d)

Cisco Unique Device Identifier Support

The ENTITY-MIB now supports the Cisco compliance effort for a Cisco unique device identifier (UDI) standard which is stored in IDPROM.

The Cisco UDI provides a unique identity for every Cisco product. The UDI is composed of three separate data elements which must be stored in the entPhysicalTable:

- Orderable product identifier (PID)—Product Identifier (PID). PID is the alphanumeric identifier used by customers to order Cisco products. Two examples include NM-1FE-TX or CISCO3745. PID is limited to 18 characters and must be stored in the entPhysicalModelName object.
- Version identifier (VID)—Version Identifier (VID). VID is the version of the PID. The VID indicates the number of times a product has versioned in ways that are reported to a customer. For example, the product identifier NM-1FE-TX may have a VID of V04. VID is limited to 3 alphanumeric characters and must be stored in the entPhysicalHardwareRev object.
- Serial number (SN)—Serial number is the 11-character identifier used to identify a specific part within a product and must be stored in the entPhysicalSerialNum object. Serial number content is defined by manufacturing part number 7018060-0000. The SN is accessed at the following website by searching on the part number 701806-0000:

  https://mco.cisco.com/servlet/mco.ecm.inbiz.inbiz

  Serial number format is defined in four fields:

  - Location (L)
  - Year (Y)
  - Workweek (W)
  - Sequential serial ID (S)

  The SN label will be represented as: LLLYYWWSSS.

Note The Version ID returns NULL for those old or existing cards whose IDPROMs do not have the Version ID field. Therefore, corresponding entPhysicalHardwareRev returns NULL for cards that do not have the Version ID field in IDPROM.