



APPENDIX **A**

Using MIBs

This chapter describes how to perform tasks on the Cisco ASR 1000 Series Routers

- [Cisco Unique Device Identifier Support, page A-1](#)
- [Cisco Redundancy Features, page A-2](#)
- [Managing Physical Entities, page A-5](#)
- [Monitoring Quality of Service, page A-25](#)
 - [CISCO-CLASS-BASED-QOS-MIB Overview, page A-25](#)
 - [Viewing QoS Configuration Settings Using the CISCO-CLASS-BASED-QOS-MIB, page A-26](#)
 - [Monitoring QoS Using the CISCO-CLASS-BASED-QOS-MIB, page A-27](#)
 - [Considerations for Processing QoS Statistics, page A-28](#)
 - [Sample QoS Applications, page A-30](#)
- [Monitoring Router Interfaces, page A-33](#)
- [Billing Customers for Traffic, page A-34](#)
- [Using IF-MIB Counters, page A-37](#)
- [Overview of SIPs and SPAs, page A-39](#)

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 three 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>

Serial number format is defined in four fields:

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

The SN label is 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.

Cisco Redundancy Features

This section describes

- [Levels of Redundancy, page A-2](#)
 - [Route Processor Redundancy, page A-3](#)
 - [Cisco Nonstop Forwarding and Stateful Switchover, page A-3](#)
- [Software Redundancy](#)
- [Verifying Cisco ASR 1000 Series Routers Redundancy, page A-4](#)
- [Related Information and Useful Links, page A-5](#)

Redundancy creates a duplication of data elements and software functions to provide an alternative in case of failure. The goal of Cisco redundancy features is to cut over without affecting the link and protocol states associated with each interface and continue packet forwarding. The state of the interfaces and subinterfaces is maintained, along with the state of line cards and various packet processing hardware.

Levels of Redundancy

This section describes the levels of redundancy supported on the Cisco ASR 1000 Series Routers and how to verify that this feature is available. Cisco ASR 1000 Series Routers support fault resistance by allowing a Cisco redundant supervisor engine (SE) to take over if the active supervisor engine fails. Redundancy prevents equipment failures from causing service outages, and supports hitless maintenance and upgrade activities. The state of the interfaces and subinterfaces are maintained along with the state of line cards and various packet processing hardware.

Redundant systems support two route processors. One acts as the active route processor while the other acts as the standby route processor.

The route processor redundancy feature provides high availability for Cisco routers by switching over to the standby route processor when one of the following conditions occur:

- Cisco IOS software failure
- Cisco ASR 1000 Series Route Processor (RP) hardware failure
- Software upgrade
- Maintenance procedure

Cisco ASR 1000 Series Routers can operate in one of two redundancy modes:

- Route Processor Redundancy (RPR) mode
- Nonstop Forwarding/Stateful Switchover (NSF/SSO) mode

In all modes, the standby RP will take over when the active RP fails.

Route Processor Redundancy

This section describes the Route Processor Redundancy (RPR) mode for the Cisco ASR 1000 Series Routers.

When the switch is powered on, RPR runs between two Cisco supervisor engines. The supervisor engine that boots first becomes the RPR active supervisor engine.

Cisco ASR 1000 Series Routers support fault resistance by allowing a redundant supervisor engine to take over if the active supervisor engine fails.

Cisco Nonstop Forwarding and Stateful Switchover

This section describes the Cisco Nonstop Forwarding and Stateful Switchover mode. With NSF/SSO, Cisco ASR 1000 Series Routers can fail over from the active to the standby route processor almost immediately while continuing to forward packets. Cisco IOS software NSF/SSO support on this platform enables immediate failover.

In networking devices running NSF/SSO, both RPs must be running the same configuration so that the standby RP is always ready to assume control following a fault on the active RP. The configuration information is synchronized from the active RP to the standby RP at startup and each time changes to the active RP configuration occur.

Following an initial synchronization between the two processors, NSF/SSO maintains RP state information between them, including forwarding information.

Cisco Nonstop Forwarding (NSF) works with the Stateful Switchover (SSO) to minimize the amount of time a network is unavailable to its users following a Route Processor (RP) fail-over in a router with dual RPs. NSF/SSO capability allows routers to detect a switchover and take the necessary actions to continue forwarding network traffic and to recover route information from peer devices.

Cisco NSF works with the Stateful Switchover (SSO) feature in Cisco IOS software to minimize the amount of time a network is unavailable to its users following a switchover. The main objective of Cisco NSF/SSO is to continue forwarding data packets along known routes while the routing protocol information is being restored following a route switchover.



Note

For detailed information about the Nonstop Forwarding feature go to:
http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fsnsf20s.html

**Note**

For detailed information about the Stateful Switchover feature go to:
http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fssso20s.html

Software Redundancy

Cisco ASR 1004 Routers having only one RP slot do not support hardware redundancy. Instead, these Routers have option of software redundancy by running two IOSD processes. IOSD can optionally be run in a redundant configuration. One IOSD instance is active and the other is maintained in a hot standby mode. State information is exchanged between the instances using the normal SSO support over IPC. Software redundancy option is not available and is deactivated if a second RP is added to the chassis. The active RP is responsible for controlling both the active and standby FP as well as all of the I/O (carrier) cards. If the active IOSD instance fails then the backup takes over and resynchronizes its state with the FP and I/O cards.

Verifying Cisco ASR 1000 Series Routers Redundancy

To display information about the active and standby supervisor engines installed in a Cisco ASR 1000 Series Routers, use the **show redundancy** command and **show redundancy states** command. For Router Processor in R0 slot, the value of Unit ID is 48, same as ASCII "0" (hex 30). The value of Unit ID is 49, ASCII "1" (hex 31), for Router Processor in R1 slot.

Example A-1 Displaying Redundancy States from Active Processor

```
R5-mcp-6ru-2#sh redundancy states
  my state = 13 -ACTIVE
  peer state = 8 -STANDBY HOT
    Mode = Duplex
    Unit ID = 48

Redundancy Mode (Operational) = sso
Redundancy Mode (Configured) = sso
Redundancy State = sso
  Maintenance Mode = Disabled
  Manual Swact = enabled
  Communications = Up

  client count = 66
  client_notification_TMR = 30000 milliseconds
  RF debug mask = 0x0

R5-mcp-6ru-2#exit
```

Example A-2 Displaying Redundancy States from Standby Processor

```
R5-mcp-6ru-2-stby#sh redundancy state
  my state = 8 -STANDBY HOT
  peer state = 13 -ACTIVE
    Mode = Duplex
    Unit ID = 49

Redundancy Mode (Operational) = sso
Redundancy Mode (Configured) = sso
Redundancy State = sso
```

```

Maintenance Mode = Disabled
Manual Swact = cannot be initiated from this the standby unit  Communications = Up

client count = 67
client_notification_TMR = 30000 milliseconds
RF debug mask = 0x0

R5-mcp-6ru-2-stby#

```

Example A-3 Displaying Redundancy States for Software Redundancy - ASR 1004

```

R5-mcp-4ru-1#sh redundancy states
my state = 13 -ACTIVE
peer state = 8 -STANDBY HOT
Mode = Duplex
Unit ID = 48

Redundancy Mode (Operational) = sso
Redundancy Mode (Configured) = sso
Redundancy State = sso
Maintenance Mode = Disabled
Manual Swact = enabled
Communications = Up

client count = 66
client_notification_TMR = 30000 milliseconds
RF debug mask = 0x0

R5-mcp-4ru-1#

```

Related Information and Useful Links

The following URLs provide access to helpful information about the Cisco redundancy feature:

- Detailed information about Cisco nonstop forwarding:
http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fnsnf20s.html
- Detailed information about the stateful switchover feature:
http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fssso20s.html
- Detailed information about the route processor redundancy feature:
http://www.cisco.com/en/US/docs/ios/12_1/12_1ex/feature/guide/12e_rpr.html

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-6](#)
 - [Determining the ifIndex Value for a Physical Port, page A-12](#)
 - [Monitoring and Configuring FRU Status, page A-12](#)
- [Generating SNMP Notifications, page A-23](#)

Purpose and Benefits

The physical entity management feature of the Cisco ASR 1000 Series Routers SNMP implementation does the following:

- 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-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.
- CISCO-ENTITY-EXT-MIB - Contains Cisco defined extensions to the entPhysicalTable of the ENTITY-MIB to provide information for entities with an entPhysicalClass value of 'module' that have a CPU, RAM/NVRAM, and/or a configuration register.
- CISCO-ENTITY-SENSOR-MIB and ENTITY-SENSOR-MIB—Contain information about entities in the entPhysicalTable with an entPhysicalClass value of 'sensor'.
- CISCO-ENTITY-VENDORTYPE-OID-MIB—Contains the object identifiers (OIDs) for all physical entities in the router.
- 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.
- The entPhysicalIsFRU indicates whether or not a physical entity is considered a Field Replaceable Unit (FRU). For an entity identified as FRU, the physical entity contains the following device-specific information:
 - entPhysicalModelName- Product Identification (PID), same as orderable part number.
 - entPhysicalHardwareRev- Version Identification (VID)
 - entPhysicalSerialNum- Serial Number (SN)
 - Cisco Unique Device Identifier (UDI)- Composed of PID, VID and SN, it provides a unique identity for all Cisco hardware products on which it has been enabled.

Performing Inventory Management

To obtain information about entities in the router, perform a MIB walk on the ENTITY-MIB entPhysicalTable.

As you examine sample 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).



Note The container is applicable if the physical entity class is capable of containing one or more removable physical entities. For example, each (empty or full) slot in a chassis is modeled as a container. All removable physical entities should be modeled within a container entity, such as field-replaceable modules, fans, or power supplies.

Sample of ENTITY-MIB entPhysicalTable Entries

The samples in this section show how information is stored in the entPhysicalTable. You can perform asset inventory by examining entPhysicalTable entries.



Note

The sample outputs and values that appear throughout this chapter are examples of data you can view when using MIBs.

The following display shows the ENTITY-MIB entPhysicalTable sample entries for a ASR1000 SIP-10 card installed in a router chassis and four SPAs inserted into the card.

ENTITY-MIB entPhysicalTable Entries

```
entPhysicalDescr.1000 = Cisco ASR1000 SPA Interface Processor 10
entPhysicalDescr.1001 = V1: VMA
entPhysicalDescr.1002 = V1: VMB
entPhysicalDescr.1003 = V1: VMC
entPhysicalDescr.1004 = V1: VMD
entPhysicalDescr.1005 = V1: VME
entPhysicalDescr.1006 = V1: VMF
entPhysicalDescr.1007 = V1: 12v
entPhysicalDescr.1008 = V1: VDD
entPhysicalDescr.1009 = V1: GP1
entPhysicalDescr.1010 = V1: GP2
entPhysicalDescr.1011 = V2: VMB
entPhysicalDescr.1012 = V2: 12v
entPhysicalDescr.1013 = V2: VDD
entPhysicalDescr.1014 = V2: GP2
entPhysicalDescr.1015 = Temp: Left
entPhysicalDescr.1016 = Temp: Center
entPhysicalDescr.1017 = Temp: Asic1
entPhysicalDescr.1018 = Temp: Right
entPhysicalDescr.1026 = CPU 0 of module 0
entPhysicalDescr.1027 = SPA Bay
entPhysicalDescr.1028 = SPA Bay
entPhysicalDescr.1029 = SPA Bay
entPhysicalDescr.1030 = SPA Bay
.....
entPhysicalVendorType.1000 = cevModuleASR1000SIP10
entPhysicalVendorType.1001 = cevSensor
entPhysicalVendorType.1002 = cevSensor
entPhysicalVendorType.1003 = cevSensor
entPhysicalVendorType.1004 = cevSensor
entPhysicalVendorType.1005 = cevSensor
entPhysicalVendorType.1006 = cevSensor
```

```

entPhysicalVendorType.1007 = cevSensor
entPhysicalVendorType.1008 = cevSensor
entPhysicalVendorType.1009 = cevSensor
entPhysicalVendorType.1010 = cevSensor
entPhysicalVendorType.1011 = cevSensor
entPhysicalVendorType.1012 = cevSensor
entPhysicalVendorType.1013 = cevSensor
entPhysicalVendorType.1014 = cevSensor
entPhysicalVendorType.1015 = cevSensorModuleDeviceTemp
entPhysicalVendorType.1016 = cevSensorModuleDeviceTemp
entPhysicalVendorType.1017 = cevSensorModuleDeviceTemp
entPhysicalVendorType.1018 = cevSensorModuleDeviceTemp
entPhysicalVendorType.1026 = cevModuleCpuType
entPhysicalVendorType.1027 = cevContainerSPABay
entPhysicalVendorType.1028 = cevContainerSPABay
entPhysicalVendorType.1029 = cevContainerSPABay
entPhysicalVendorType.1030 = cevContainerSPABay
. . . .

```

where **entPhysicalVendorType** identifies the unique vendor-specific hardware type of the physical entity.

```

entPhysicalContainedIn.1000 = 2
entPhysicalContainedIn.1001 = 1000
entPhysicalContainedIn.1002 = 1000
entPhysicalContainedIn.1003 = 1000
entPhysicalContainedIn.1004 = 1000
entPhysicalContainedIn.1005 = 1000
entPhysicalContainedIn.1006 = 1000
entPhysicalContainedIn.1007 = 1000
entPhysicalContainedIn.1008 = 1000
entPhysicalContainedIn.1009 = 1000
entPhysicalContainedIn.1010 = 1000
entPhysicalContainedIn.1011 = 1000
entPhysicalContainedIn.1012 = 1000
entPhysicalContainedIn.1013 = 1000
entPhysicalContainedIn.1014 = 1000
entPhysicalContainedIn.1015 = 1000
entPhysicalContainedIn.1016 = 1000
entPhysicalContainedIn.1017 = 1000
entPhysicalContainedIn.1018 = 1000
entPhysicalContainedIn.1026 = 1000
entPhysicalContainedIn.1027 = 1000
entPhysicalContainedIn.1028 = 1000
entPhysicalContainedIn.1029 = 1000
entPhysicalContainedIn.1030 = 1000

```

where **entPhysicalContainedIn** indicates the entPhysicalIndex of a component's parent entity.

```

entPhysicalClass.1000 = module(9)
entPhysicalClass.1001 = sensor(8)
entPhysicalClass.1002 = sensor(8)
entPhysicalClass.1003 = sensor(8)
entPhysicalClass.1004 = sensor(8)
entPhysicalClass.1005 = sensor(8)
entPhysicalClass.1006 = sensor(8)
entPhysicalClass.1007 = sensor(8)
entPhysicalClass.1008 = sensor(8)
entPhysicalClass.1009 = sensor(8)
entPhysicalClass.1010 = sensor(8)
entPhysicalClass.1011 = sensor(8)
entPhysicalClass.1012 = sensor(8)
entPhysicalClass.1013 = sensor(8)

```

```

entPhysicalClass.1014 = sensor(8)
entPhysicalClass.1015 = sensor(8)
entPhysicalClass.1016 = sensor(8)
entPhysicalClass.1017 = sensor(8)
entPhysicalClass.1018 = sensor(8)
entPhysicalClass.1026 = other(1)
entPhysicalClass.1027 = container(5)
entPhysicalClass.1028 = container(5)
entPhysicalClass.1029 = container(5)
entPhysicalClass.1030 = container(5)

```

where **entPhysicalClass** indicates the general type of hardware device.

```

entPhysicalParentRelPos.1000 = 0
entPhysicalParentRelPos.1001 = 0
entPhysicalParentRelPos.1002 = 1
entPhysicalParentRelPos.1003 = 2
entPhysicalParentRelPos.1004 = 3
entPhysicalParentRelPos.1005 = 4
entPhysicalParentRelPos.1006 = 5
entPhysicalParentRelPos.1007 = 6
entPhysicalParentRelPos.1008 = 7
entPhysicalParentRelPos.1009 = 8
entPhysicalParentRelPos.1010 = 9
entPhysicalParentRelPos.1011 = 10
entPhysicalParentRelPos.1012 = 11
entPhysicalParentRelPos.1013 = 12
entPhysicalParentRelPos.1014 = 13
entPhysicalParentRelPos.1015 = 14
entPhysicalParentRelPos.1016 = 15
entPhysicalParentRelPos.1017 = 16
entPhysicalParentRelPos.1018 = 17
entPhysicalParentRelPos.1026 = 0
entPhysicalParentRelPos.1027 = 0
entPhysicalParentRelPos.1028 = 1
entPhysicalParentRelPos.1029 = 2
entPhysicalParentRelPos.1030 = 3

```

where **entPhysicalParentRelPos** indicates the relative position of this *child* among the other entities.

```

entPhysicalName.1000 = module 0
entPhysicalName.1001 = V1: VMA 0/0
entPhysicalName.1002 = V1: VMB 0/1
entPhysicalName.1003 = V1: VMC 0/2
entPhysicalName.1004 = V1: VMD 0/3
entPhysicalName.1005 = V1: VME 0/4
entPhysicalName.1006 = V1: VMF 0/5
entPhysicalName.1007 = V1: 12v 0/6
entPhysicalName.1008 = V1: VDD 0/7
entPhysicalName.1009 = V1: GP1 0/8
entPhysicalName.1010 = V1: GP2 0/9
entPhysicalName.1011 = V2: VMB 0/10
entPhysicalName.1012 = V2: 12v 0/11
entPhysicalName.1013 = V2: VDD 0/12
entPhysicalName.1014 = V2: GP2 0/13
entPhysicalName.1015 = Temp: Left 0/14
entPhysicalName.1016 = Temp: Center 0/15
entPhysicalName.1017 = Temp: Asic1 0/16
entPhysicalName.1018 = Temp: Right 0/17
entPhysicalName.1026 = cpu 0/0
entPhysicalName.1027 = subslot 0/0
entPhysicalName.1028 = subslot 0/1

```

```
entPhysicalName.1029 = subslot 0/2
entPhysicalName.1030 = subslot 0/3
```

where **entPhysicalName** provides the textual name of the physical entity.

```
entPhysicalHardwareRev.1000 = V00
entPhysicalHardwareRev.1001 =
entPhysicalHardwareRev.1002 =
entPhysicalHardwareRev.1003 =
entPhysicalHardwareRev.1004 =
entPhysicalHardwareRev.1005 =
entPhysicalHardwareRev.1006 =
entPhysicalHardwareRev.1007 =
entPhysicalHardwareRev.1008 =
entPhysicalHardwareRev.1009 =
entPhysicalHardwareRev.1010 =
entPhysicalHardwareRev.1011 =
entPhysicalHardwareRev.1012 =
entPhysicalHardwareRev.1013 =
entPhysicalHardwareRev.1014 =
entPhysicalHardwareRev.1015 =
entPhysicalHardwareRev.1016 =
entPhysicalHardwareRev.1017 =
entPhysicalHardwareRev.1018 =
entPhysicalHardwareRev.1026 =
entPhysicalHardwareRev.1027 =
entPhysicalHardwareRev.1028 =
entPhysicalHardwareRev.1029 =
entPhysicalHardwareRev.1030 =
```

where **entPhysicalHardware** provides the vendor-specific hardware revision number (string) for the physical entity.

```
entPhysicalSerialNum.1000 = JAB11090506
entPhysicalSerialNum.1001 =
entPhysicalSerialNum.1002 =
entPhysicalSerialNum.1003 =
entPhysicalSerialNum.1004 =
entPhysicalSerialNum.1005 =
entPhysicalSerialNum.1006 =
entPhysicalSerialNum.1007 =
entPhysicalSerialNum.1008 =
entPhysicalSerialNum.1009 =
entPhysicalSerialNum.1010 =
entPhysicalSerialNum.1011 =
entPhysicalSerialNum.1012 =
entPhysicalSerialNum.1013 =
entPhysicalSerialNum.1014 =
entPhysicalSerialNum.1015 =
entPhysicalSerialNum.1016 =
entPhysicalSerialNum.1017 =
entPhysicalSerialNum.1018 =
entPhysicalSerialNum.1026 =
entPhysicalSerialNum.1027 =
entPhysicalSerialNum.1028 =
entPhysicalSerialNum.1029 =
entPhysicalSerialNum.1030 =
```

where **entPhysicalSerialNumber** provides the vendor-specific serial number (string) for the physical entity.

```
entPhysicalMfgName.1000 = Cisco Systems Inc
```

```
entPhysicalMfgName.1001 =
entPhysicalMfgName.1002 =
entPhysicalMfgName.1003 =
entPhysicalMfgName.1004 =
entPhysicalMfgName.1005 =
entPhysicalMfgName.1006 =
entPhysicalMfgName.1007 =
entPhysicalMfgName.1008 =
entPhysicalMfgName.1009 =
entPhysicalMfgName.1010 =
entPhysicalMfgName.1011 =
entPhysicalMfgName.1012 =
entPhysicalMfgName.1013 =
entPhysicalMfgName.1014 =
entPhysicalMfgName.1015 =
entPhysicalMfgName.1016 =
entPhysicalMfgName.1017 =
entPhysicalMfgName.1018 =
entPhysicalMfgName.1026 =
entPhysicalMfgName.1027 =
entPhysicalMfgName.1028 =
entPhysicalMfgName.1029 =
entPhysicalMfgName.1030 =
```

where **entPhysicalMfgName** provides the manufacturer's name for the physical component.

```
entPhysicalModelName.1000 = ASR1000-SIP10
entPhysicalModelName.1001 =
entPhysicalModelName.1002 =
entPhysicalModelName.1003 =
entPhysicalModelName.1004 =
entPhysicalModelName.1005 =
entPhysicalModelName.1006 =
entPhysicalModelName.1007 =
entPhysicalModelName.1008 =
entPhysicalModelName.1009 =
entPhysicalModelName.1010 =
entPhysicalModelName.1011 =
entPhysicalModelName.1012 =
entPhysicalModelName.1013 =
entPhysicalModelName.1014 =
entPhysicalModelName.1015 =
entPhysicalModelName.1016 =
entPhysicalModelName.1017 =
entPhysicalModelName.1018 =
entPhysicalModelName.1026 =
entPhysicalModelName.1027 =
entPhysicalModelName.1028 =
entPhysicalModelName.1029 =
entPhysicalModelName.1030 =
```

where **entPhysicalModelName** provides the vendor-specific model name string for the physical component.

```
entPhysicalIsFRU.1000 = true(1)
entPhysicalIsFRU.1001 = false(2)
entPhysicalIsFRU.1002 = false(2)
entPhysicalIsFRU.1003 = false(2)
entPhysicalIsFRU.1004 = false(2)
entPhysicalIsFRU.1005 = false(2)
entPhysicalIsFRU.1006 = false(2)
entPhysicalIsFRU.1007 = false(2)
```

```

entPhysicalIsFRU.1008 = false(2)
entPhysicalIsFRU.1009 = false(2)
entPhysicalIsFRU.1010 = false(2)
entPhysicalIsFRU.1011 = false(2)
entPhysicalIsFRU.1012 = false(2)
entPhysicalIsFRU.1013 = false(2)
entPhysicalIsFRU.1014 = false(2)
entPhysicalIsFRU.1015 = false(2)
entPhysicalIsFRU.1016 = false(2)
entPhysicalIsFRU.1017 = false(2)
entPhysicalIsFRU.1018 = false(2)
entPhysicalIsFRU.1026 = false(2)
entPhysicalIsFRU.1027 = false(2)
entPhysicalIsFRU.1028 = false(2)
entPhysicalIsFRU.1029 = false(2)
entPhysicalIsFRU.1030 = false(2)

```

where **entPhysicalIsFRU** indicates whether or not this physical entity is considered a field replaceable unit (FRU).

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 SPA ports, the **entPhysicalContainedIn** = 1280 (the **entPhysicalIndex** of the SPA card).
- Each chassis slot and line card port has a different **entPhysicalParentRelPos** to show its relative position within the parent object.

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**. 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.)

```
entAliasMappingIdentifier.1813.0 = ifIndex.4
```

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.

Figure A-1 shows a **cefcModuleTable** entry for a SIP card whose **entPhysicalIndex** is 1000.

Figure A-1 Sample cefcModuleTable Entry

```

cefcModuleAdminStatus.1000 = enabled(1)
cefcModuleOperStatus.1000 = ok(2)
cefcModuleResetReason.1000 = unknown(1)
cefcModuleStatusLastChangeTime.1000 =
15865

```

See the “[FRU Status Changes](#)” section on page A-24 for information about how the router generates notifications to indicate changes in FRU status.

Using ENTITY-ALARM-MIB to Monitor Entity Alarms

ENTITY-MIB

The Entity physical table contains information for managing physical entities on the router. It also organizes the entities into a containment tree that depicts their hierarchy, and relationship with each other. Refer to the [Appendix A, “Entity Containment Tree”](#) section for the entity hierarchy. The following sample output contains the information for the ASR1002 AC power supply in power supply bay 0:

```

ptolemy-265->getmany -v2c 9.0.0.56 public entityMIB | grep "\.4 "
entPhysicalDescr.4 = Cisco ASR1002 AC Power Supply
entPhysicalVendorType.4 = cevPowerSupplyASR1002AC
entPhysicalContainedIn.4 = 3
entPhysicalClass.4 = powerSupply(6)
entPhysicalParentRelPos.4 = 0
entPhysicalName.4 = Power Supply Module 0
entPhysicalHardwareRev.4 = V01
entPhysicalFirmwareRev.4 =
entPhysicalSoftwareRev.4 =
entPhysicalSerialNum.4 = ART1132U00C
entPhysicalMfgName.4 =
entPhysicalModelName.4 = ASR1002-PWR-AC
entPhysicalAlias.4 =
entPhysicalAssetID.4 =
entPhysicalIsFRU.4 = true(1)
entPhysicalMfgDate.4 = 00 00 00 00 00 00 00 00
entPhysicalUri.4 = URN:CLEI:COUPACJBAA
entPhysicalChildIndex.3.4 = 4

```

For more information on this MIB, refer to [ENTITY-MIB \(RFC 4133\)](#), page 3-87.

CISCO-ENTITY-ALARM-MIB

CISCO-ENTITY-ALARM-MIB supports the monitoring of alarms generated by physical entities contained by the system, including chassis, slots, modules, ports, power supplies, etc. In order to monitor alarms generated by a physical entity, it must be represented by a row in the entPhysicalTable.

For more information on this MIB, refer to [CISCO-ENTITY-ALARM-MIB](#), page 3-29.

Alarm Description Map Table

For each type of entity (represented by entPhysicalVendorType OID), this table contains a mapping between a unique ceAlarmDescrIndex and entPhysicalVendorType OID.

The ceAlarmDescrMapEntry is indexed by the CeAlarmDescrMapEntry.



Note

The mapping between the ceAlarmDescrIndex and entPhysicalVendorType OID will exist only if the type of entity supports alarms monitoring, and it is in the device since device boot-up.

The following are the sample output:

```
ptolemy-218->getmany -v2c 9.0.0.56 public ceAlarmDescrMapTable
ceAlarmDescrVendorType.1 = cevPortCT3
ceAlarmDescrVendorType.2 = cevPortT1E1
ceAlarmDescrVendorType.3 = cevPortT3E3
ceAlarmDescrVendorType.4 = cevContainerSFP
ceAlarmDescrVendorType.5 = cevContainerASR1000RPSlot
ceAlarmDescrVendorType.6 = cevContainerASR1000FPSlot
ceAlarmDescrVendorType.7 = cevContainerASR1000CCSlot
ceAlarmDescrVendorType.8 = cevContainerASR1000PowerSupplyBay
ceAlarmDescrVendorType.9 = cevSensorModuleDeviceTemp
ceAlarmDescrVendorType.10 = cevSensorModuleDeviceVoltage
ceAlarmDescrVendorType.11 = cevSensorModuleDeviceCurrent
ceAlarmDescrVendorType.12 = cevSensor
ceAlarmDescrVendorType.13 = cevModuleASR1002RP1
ceAlarmDescrVendorType.14 = cevPortUSB
ceAlarmDescrVendorType.15 = cevPortGe
ceAlarmDescrVendorType.16 = cevModuleASR1000ESP10
ceAlarmDescrVendorType.17 = cevModuleASR1002SIP10
ceAlarmDescrVendorType.18 = cevContainerSPABay
ceAlarmDescrVendorType.19 = cevPowerSupplyASR1002AC
ceAlarmDescrVendorType.20 = cevModuleASR1002Spa4pGe
```

The temperature sensor in ASR1000 modules (RP, FP, CC, and PEM) contains cevSensorModuleDeviceTemp as entPhysicalVendorType OID. From the above sample output, the index (ceAlarmDescrIndex) 9 is mapped to cevSensorModuleDeviceTemp, and the index 19 is mapped to the AER10002 power supply which has cevPowerSupplyASR1002AC as entity physical vendor type OID.



Note

SPA is not included in ALL ASR1000 modules. It has its own vendor type OID defined for its sensor.



Note

The generic vendor OID, cevSensor, is used in case the ASR1000 snmp agent is not able to determine the sensor type.

Alarm Description Table

The Alarm Description Table contains a description for each alarm type, defined by each vendor type employed by the system. Each alarm description entry (ceAlarmDescrEntry) is indexed by ceAlarmDescrIndex and ceAlarmDescrAlarmType.

The following is the sample output for all alarm types defined for all temperature type of entity in the ASR1000 modules. The index 9 is obtained from the ceAlarmDescrMapTable in the previous section:

```
ptolemy-225->getmany -v2c 9.0.0.56 public ceAlarmDescrTable | grep "\.9\."
ceAlarmDescrSeverity.9.0 = 1
```

```

ceAlarmDescrSeverity.9.1 = 1
ceAlarmDescrSeverity.9.2 = 1
ceAlarmDescrSeverity.9.3 = 2
ceAlarmDescrSeverity.9.4 = 3
ceAlarmDescrSeverity.9.5 = 1
ceAlarmDescrSeverity.9.6 = 1
ceAlarmDescrSeverity.9.7 = 2
ceAlarmDescrSeverity.9.8 = 3
ceAlarmDescrText.9.0 = Faulty Temperature Sensor
ceAlarmDescrText.9.1 = Temp Above Normal (Shutdown)
ceAlarmDescrText.9.2 = Temp Above Normal
ceAlarmDescrText.9.3 = Temp Above Normal
ceAlarmDescrText.9.4 = Temp Above Normal
ceAlarmDescrText.9.5 = Temp Below Normal (Shutdown)
ceAlarmDescrText.9.6 = Temp Below Normal
ceAlarmDescrText.9.7 = Temp Below Normal
ceAlarmDescrText.9.8 = Temp Below Normal

```

Refer to the Bellcore Technical Reference TR-NWT-000474 Issue 4, December 1993, OTGR Section 4. Network Maintenance: Alarm and Control - Network Element. The severity is defined as follows:

- critical(1)
- major(2)
- minor(3)
- info(4)

The following is the list of alarms defined for the sensor:

```

Alarm type 0 is for faulty sensor
Alarm type 1 is for crossing the shutdown threshold (above normal range).
Alarm type 2 is for crossing the critical threshold (above normal range).
Alarm type 3 is for crossing the major threshold (above normal range).
Alarm type 4 is for crossing the minor threshold (above normal range).
Alarm type 5 is for crossing the shutdown threshold (below normal range).
Alarm type 6 is for crossing the critical threshold (below normal range).
Alarm type 7 is for crossing the major threshold (below normal range).
Alarm type 8 is for crossing the minor threshold (below normal range).

```

These alarm types are defined for all sensor physical entity type. The only difference is that different sensor physical type have different ceAlarmDescrText. The temperature sensor has "TEMP" and the voltage sensor has "Volt" in the alarm description text.

The following is the sample output of all alarm types. It is defined for the ASR1002 AC power supply which has cevPowerSupplyASR1002AC as vendor type OID and is mapped to the ceAlarmDescrIndex 19.

```

ptolemy-237->getmany -v2c 9.0.0.56 public ceAlarmDescrTable | grep "\.19\"
ceAlarmDescrSeverity.19.0 = 1
ceAlarmDescrSeverity.19.1 = 1
ceAlarmDescrSeverity.19.2 = 1
ceAlarmDescrSeverity.19.3 = 2
ceAlarmDescrSeverity.19.4 = 2
ceAlarmDescrSeverity.19.5 = 2
ceAlarmDescrText.19.0 = Power Supply Failure
ceAlarmDescrText.19.1 = All Fans Failed
ceAlarmDescrText.19.2 = Multiple Fan Failures
ceAlarmDescrText.19.3 = Fan 0 Failure
ceAlarmDescrText.19.4 = Fan 1 Failure
ceAlarmDescrText.19.5 = Fan 2 Failure

```

Alarm Table

The Alarm Table specifies alarm control and status information related to each physical entity contained by the system. The table includes the alarms currently being asserted by each physical entity that is capable of generating alarms. Each physical entity in entity physical table that is capable of generating alarms has an entry in this table. The alarm entry (ceAlarmEntry) is indexed by the entity physical index (entPhysicalIndex). The following is a list of MIB objects in the alarm entry:

- **ceAlarmFilterProfile**
The alarm filter profile object contains an integer value that uniquely identifies an alarm filter profile associated with the corresponding physical entity. An alarm filter profile controls which alarm types the agent will monitor and signal for the corresponding physical entity. The default value of this object is 0, the agent monitors and signals all alarms associated with the corresponding physical entity.
- **ceAlarmSeverity**
This object specifies the highest severity alarm currently being asserted by the corresponding physical entity.
A value of '0' indicates that the corresponding physical entity is not currently asserting any alarms.
- **ceAlarmList**
This object specifies those alarms currently being asserted by the corresponding physical entity. If an alarm is being asserted by the physical entity, then the corresponding bit in the alarm list is set to a one. The alarm list is defined as octet string and its size ranges from 0 to 32.
 - If the physical entity is not currently asserting any alarms, then the list will have a length of zero, otherwise it will have a length of 32.
 - An OCTET STRING represents an alarm list, in which each bit represents an alarm type:

octet 1:

```

  7 6 5 4 3 2 1 0
+-----+
|       |
+-----+
| | | | | | | |
| | | | | | +- Alarm type 0
| | | | | | +--- Alarm type 1
| | | | | +----- Alarm type 2
| | | | +----- Alarm type 3
| | | +----- Alarm type 4
| | +----- Alarm type 5
| +----- Alarm type 6
+----- Alarm type 7

```

octet 2:

```

  7 6 5 4 3 2 1 0
+-----+
|       |
+-----+
| | | | | | | |
| | | | | | +- Alarm type 8
| | | | | | +--- Alarm type 9
| | | | | +----- Alarm type 10
| | | | +----- Alarm type 11
| | | +----- Alarm type 12
| | +----- Alarm type 13
| +----- Alarm type 14
+----- Alarm type 15

```

octet xx

octet 32:

```

 7 6 5 4 3 2 1 0
 +---+---+---+---+
 |                                     |
 +---+---+---+---+
 | | | | | | | |
 | | | | | | | +- Alarm type 248
 | | | | | | | +--- Alarm type 249
 | | | | | | | +----- Alarm type 250
 | | | | | | | +----- Alarm type 251
 | | | | | | | +----- Alarm type 252
 | | | | | | | +----- Alarm type 253
 | | | | | | | +----- Alarm type 254
 +----- Alarm type 255

```

From the entity physical table (entPhysicalTable in ENTITY-MIB), we understand that the ASR1002 AC power supply in power supply bay 0 has 4 as entPhysicalIndex .

The following are the sample output of alarm list for the power supply in PS bay 0:

```

ptolemy-248->getone -v2c 9.0.0.56 public ceAlarmList.4
ceAlarmList.4 =
09 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00

```

octet 1: 09

```

 7 6 5 4 3 2 1 0
 +---+---+---+---+
 0 0 0 0 1 0 0 1
 +---+---+---+---+
 | | | | | | | |
 | | | | | | | +- Alarm type 0
 | | | | | | | +--- Alarm type 1
 | | | | | | | +----- Alarm type 2
 | | | | | | | +----- Alarm type 3
 | | | | | | | +----- Alarm type 4
 | | | | | | | +----- Alarm type 5
 | | | | | | | +----- Alarm type 6
 +----- Alarm type 7

```

From the sample output in the Alarm Description Table section and the alarm mapping table, the ASR1002 AC power supply in the bay 0 has the following alarms asserted:

Alarm type 0 : Power Supply Failure

Alarm type 3 : Fan 0 Failure

As for the ASR1002 AC power supply in bay 1, which has 14 as entPhysicalIndex:

```

ptolemy-247->getone -v2c 9.0.0.56 public ceAlarmList.14
ceAlarmList.14 =

```

Because the length of alarm list returned for the power supply in bay 1 is 0, there is no alarm asserted for the power supply in bay 1.

The following is the output of "show facility-alarm status" CLI command; it displays all alarms currently asserted in the device:

```

R5-mcp-2ru-1#sh facility-alarm status
System Totals Critical: 2 Major: 1 Minor: 0
Source          Severity      Description [Index]

```

```

-----
Cisco ASR1002 AC Power Sup CRITICAL      Power Supply Failure [0]
Cisco ASR1002 AC Power Sup MAJOR         Fan 0 Failure [3]
xcvr container 0/0/1         INFO           Transceiver Missing [0]
xcvr container 0/0/2         CRITICAL      Transceiver Missing - Link Down [1]
xcvr container 0/0/3         INFO           Transceiver Missing [0]

```

Alarm History Table

The Alarm History Table, `ceAlarmHistTable`, contains history of alarms both asserted and cleared generated by the agent. The `ceAlarmHistTableSize` is used to control the size of the alarm history table. A value of 0 prevents any history from being retained in this table. If the capacity of the `ceAlarmHistTable` has reached the value specified by this object, then the agent deletes the oldest entry in order to accommodate a new entry.

The `ceAlarmHistLastIndex` object contains the last index corresponding to the last entry added to the table by the snmp agent in the device. If the management client uses notifications listed in the [Appendix A, "Alarm Notifications"](#) defined in `CISCO-ENTITY-ALARM-MIB` module, then it can poll this object to determine whether it has missed a notification sent by the agent.

The following is a list of MIB objects defined in the `ceAlarmHistEntry`, which is indexed by the `ceAlarmHistIndex`:

- **ceAlarmHistIndex**
This is an integer value uniquely identifying the entry in the table. The value of this object starts at '1' and monotonically increases for each alarm (asserted or cleared) added to the alarm history table. If the value of this object is '4294967295', it will be reset to '1', upon monitoring the next alarm condition transition.
- **ceAlarmHistType**
This object indicates that the entry is added as a result of of an alarm being asserted or cleared.
- **ceAlarmHistEntPhysicalIndex**
This object contains the `entPhysicalIndex` of the physical entity that generated the alarm.
- **ceAlarmHistAlarmType**
This object specifies the type of alarm generated.
- **ceAlarmHistSeverity**
This object specifies the severity of the alarm generated.
- **ceAlarmHistTimeStamp**
This object specifies the value of the `sysUpTime` object at the time the alarm is generated.

Example A-4 Displaying Sample Output for the Alarm History

```

ptolemy-257->getnext -v2c 9.0.0.56 public ceAlarmHistory
ceAlarmHistTableSize.0 = 200 → the size of alarm history table
ptolemy-258->getnext -v2c 9.0.0.56 public ceAlarmHistTableSize.0
ceAlarmHistLastIndex.0 = 21 → the index for the last alarm added

```

Example A-5 Displaying the Last Alarm Action (asserted or cleared) Added to the Alarm History Table

```

ptolemy-259->getmany -v2c 9.0.0.56 public ceAlarmHistTable | grep "\.21 "
ceAlarmHistType.21 = cleared(2) → alarm cleared
ceAlarmHistEntPhysicalIndex.21=4 → it is for physical entity indexed by 4
ceAlarmHistAlarmType.21 = 3 → alarm type is 3
ceAlarmHistSeverity.21 = major(2) → the alarm severity is major(2)
ceAlarmHistTimeStamp.21 = 7506193

```

At this point, the EMS application should already have all information regarding the physical entity and the entity alarm type defined for the physical entity.

Example A-6 Displaying the Physical Entity That has Value 4 as entPhysicalIndex

```
entPhysicalDescr.4 = Cisco ASR1002 AC Power Supply
entPhysicalVendorType.4 = cevPowerSupplyASR1002AC
entPhysicalContainedIn.4 = 3
entPhysicalClass.4 = powerSupply(6)
entPhysicalParentRelPos.4 = 0
entPhysicalName.4 = Power Supply Module 0
entPhysicalHardwareRev.4 = V01
entPhysicalFirmwareRev.4 =
entPhysicalSoftwareRev.4 =
entPhysicalSerialNum.4 = ART1132U00C
entPhysicalMfgName.4 =
entPhysicalModelName.4 = ASR1002-PWR-AC
```

Example A-7 Displaying the Alarm Type Defined for cevPowerSupplyASR1002AC

```
ceAlarmDescrSeverity.19.0 = 1
ceAlarmDescrSeverity.19.1 = 1
ceAlarmDescrSeverity.19.2 = 1
ceAlarmDescrSeverity.19.3 = 2
ceAlarmDescrSeverity.19.4 = 2
ceAlarmDescrSeverity.19.5 = 2
ceAlarmDescrText.19.0 = Power Supply Failure
ceAlarmDescrText.19.1 = All Fans Failed
ceAlarmDescrText.19.2 = Multiple Fan Failures
ceAlarmDescrText.19.3 = Fan 0 Failure
ceAlarmDescrText.19.4 = Fan 1 Failure
ceAlarmDescrText.19.5 = Fan 2 Failure
```

From the alarm type defined for cevPowerSupplyASR1002AC, the application can easily interpret the last entry in the alarm history table as : Fan 0 Failure Alarm is Cleared for Cisco ASR1002 AC Power Supply in power supply bay 0.

Alarm Notifications

CISCO-ENTITY-ALARM-MIB supports the alarm asserted (ceAlarmAsserted) and alarm cleared (ceAlarmCleared) notifications. The notification can be enabled by setting the ceAlarmNotifiesEnable object through the snmp SET. The ceAlarmNotifiesEnable contains the severity level of the alarms notification or the value 0:

```
severity 1: critical      Service affecting Condition
severity 2: major        Immediate action needed
severity 3: minor        Minor warning conditions
severity 4: informational Informational messages
```

The severity 4 will enable notification for all severity level.

The severity 3 will enable notifications for severity 1, 2, and 3.

The severity 2 will enable notifications for severity 1 and 2.

The severity 1 will enable notifications for severity 1 only.

The value of 0 will disable the alarm notification.


```

|
| +-9004 (cevSensorModuleDeviceVoltage) : V1: VMD F0/3 : sensor
| |
| +-9005 (cevSensorModuleDeviceVoltage) : V1: VME F0/4 : sensor
| |
| +-9006 (cevSensorModuleDeviceVoltage) : V1: 12v F0/5 : sensor
| |
| +-9007 (cevSensorModuleDeviceVoltage) : V1: VDD F0/6 : sensor
| |
| +-9008 (cevSensorModuleDeviceVoltage) : V1: GP1 F0/7 : sensor
| |
| +-9009 (cevSensorModuleDeviceVoltage) : V2: VMA F0/8 : sensor
| |
| +-9010 (cevSensorModuleDeviceVoltage) : V2: VMB F0/9 : sensor
| |
| +-9011 (cevSensorModuleDeviceVoltage) : V2: VMC F0/10 : sensor
| |
| +-9012 (cevSensorModuleDeviceVoltage) : V2: VMD F0/11 : sensor
| |
| +-9013 (cevSensorModuleDeviceVoltage) : V2: VME F0/12 : sensor
| |
| +-9014 (cevSensorModuleDeviceVoltage) : V2: VMF F0/13 : sensor
| |
| +-9015 (cevSensorModuleDeviceVoltage) : V2: 12v F0/14 : sensor
| |
| +-9016 (cevSensorModuleDeviceVoltage) : V2: VDD F0/15 : sensor
| |
| +-9017 (cevSensorModuleDeviceVoltage) : V2: GP1 F0/16 : sensor
| |
| +-9018 (cevSensorModuleDeviceTemp) : Temp: Inlet F0/17 : sensor
| |
| +-9019 (cevSensorModuleDeviceTemp) : Temp: Asic1 F0/18 : sensor
| |
| +-9020 (cevSensorModuleDeviceTemp) : Temp: Exhaust1 F0/19 : sensor
| |
| +-9021 (cevSensorModuleDeviceTemp) : Temp: Exhaust2 F0/20 : sensor
| |
| \-9022 (cevSensorModuleDeviceTemp) : Temp: Asic2 F0/21 : sensor
|
+-3 (cevContainerASR1000PowerSupplyBay) : Power Supply Bay 0 : container
|
| \-4 (cevPowerSupplyASR1002AC) : Power Supply Module 0 : powerSupply
| |
| +-5 (cevSensorModuleDeviceCurrent) : PEM Iout P0/0 : sensor
| |
| +-6 (cevSensorModuleDeviceVoltage) : PEM Vout P0/1 : sensor
| |
| +-7 (cevSensorModuleDeviceVoltage) : PEM Vin P0/2 : sensor
| |
| +-8 (cevSensorModuleDeviceTemp) : Temp: PEM P0/3 : sensor
| |
| \-9 (cevSensorModuleDeviceTemp) : Temp: FC P0/4 : sensor
|
+-13 (cevContainerASR1000PowerSupplyBay) : Power Supply Bay 1 : container
|
| \-14 (cevPowerSupplyASR1002AC) : Power Supply Module 1 : powerSupply
| |
| +-15 (cevSensorModuleDeviceCurrent) : PEM Iout P1/0 : sensor
| |
| +-16 (cevSensorModuleDeviceVoltage) : PEM Vout P1/1 : sensor
| |
| +-17 (cevSensorModuleDeviceVoltage) : PEM Vin P1/2 : sensor
| |
| +-18 (cevSensorModuleDeviceTemp) : Temp: PEM P1/3 : sensor

```

```

|
|         \-19 (cevSensorModuleDeviceTemp) : Temp: FC P1/4 : sensor
+-1000 (cevModuleASR1002SIP10) : module 0 : module
|
|     +-1001 (cevSensorModuleDeviceVoltage) : V1: VMA 0/0 : sensor
|     +-1002 (cevSensorModuleDeviceVoltage) : V1: VMB 0/1 : sensor
|     +-1003 (cevSensorModuleDeviceVoltage) : V1: VMC 0/2 : sensor
|     +-1004 (cevSensorModuleDeviceVoltage) : V1: VMD 0/3 : sensor
|     +-1005 (cevSensorModuleDeviceVoltage) : V1: VME 0/4 : sensor
|     +-1006 (cevSensorModuleDeviceVoltage) : V1: VMF 0/5 : sensor
|     +-1007 (cevSensorModuleDeviceVoltage) : V1: 12v 0/6 : sensor
|     +-1008 (cevSensorModuleDeviceVoltage) : V1: VDD 0/7 : sensor
|     +-1009 (cevSensorModuleDeviceVoltage) : V1: GP1 0/8 : sensor
|     +-1010 (cevSensorModuleDeviceVoltage) : V1: GP2 0/9 : sensor
|     +-1011 (cevSensorModuleDeviceVoltage) : V2: VMB 0/10 : sensor
|     +-1012 (cevSensorModuleDeviceVoltage) : V2: 12v 0/11 : sensor
|     +-1013 (cevSensorModuleDeviceVoltage) : V2: VDD 0/12 : sensor
|     +-1014 (cevSensorModuleDeviceVoltage) : V2: GP2 0/13 : sensor
|     +-1015 (cevSensorModuleDeviceTemp) : Temp: Left 0/14 : sensor
|     +-1016 (cevSensorModuleDeviceTemp) : Temp: Center 0/15 : sensor
|     +-1017 (cevSensorModuleDeviceTemp) : Temp: Asic1 0/16 : sensor
|     +-1018 (cevSensorModuleDeviceTemp) : Temp: Right 0/17 : sensor
|
|     +-1026 (cevModuleCpuType) : cpu 0/0 : other
|     +-1027 (cevContainerSPABay) : subslot 0/1 : container
|     +-1028 (cevContainerSPABay) : subslot 0/2 : container
|     +-1029 (cevContainerSPABay) : subslot 0/3 : container
|
|     \-1040 (cevModuleASR1002Spa4pGe) : SPA subslot 0/0 : module
|         +-1066 (cevSensorModuleDeviceTemp) : subslot 0/0 temperature Sensor 0
|         +-1067 (cevSensorModuleDeviceTemp) : subslot 0/0 temperature Sensor 1
|         +-1091 (cevContainerSFP) : subslot 0/0 transceiver container 0 : cont+
|             |
|             \-1092 (cevSFP1000BaseT) : subslot 0/0 transceiver 0 : module
|                 |
|                 \-1093 (cevPortGe) : GigabitEthernet0/0/0 : port
|
|     +-1103 (cevContainerSFP) : subslot 0/0 transceiver container 1 : cont+
|
|     +-1115 (cevContainerSFP) : subslot 0/0 transceiver container 2 : cont+

```

```

|
|          \-1127 (cevContainerSFP) : subslot 0/0 transceiver container 3 : cont+
|
|-7000 (cevModuleASR1002RP1) : module R0 : module
|
|   +-7001 (cevSensorModuleDeviceVoltage) : V1: VMA R0/0 : sensor
|   |
|   +-7002 (cevSensorModuleDeviceVoltage) : V1: VMB R0/1 : sensor
|   |
|   +-7003 (cevSensorModuleDeviceVoltage) : V1: VMC R0/2 : sensor
|   |
|   +-7004 (cevSensorModuleDeviceVoltage) : V1: VMD R0/3 : sensor
|   |
|   +-7005 (cevSensorModuleDeviceVoltage) : V1: VME R0/4 : sensor
|   |
|   +-7006 (cevSensorModuleDeviceVoltage) : V1: VMF R0/5 : sensor
|   |
|   +-7007 (cevSensorModuleDeviceVoltage) : V1: 12v R0/6 : sensor
|   |
|   +-7008 (cevSensorModuleDeviceVoltage) : V1: VDD R0/7 : sensor
|   |
|   +-7009 (cevSensorModuleDeviceVoltage) : V1: GP1 R0/8 : sensor
|   |
|   +-7010 (cevSensorModuleDeviceVoltage) : V1: GP2 R0/9 : sensor
|   |
|   +-7011 (cevSensorModuleDeviceTemp) : Temp: CPU R0/10 : sensor
|   |
|   +-7012 (cevSensorModuleDeviceTemp) : Temp: Outlet R0/11 : sensor
|   |
|   +-7013 (cevSensorModuleDeviceTemp) : Temp: Inlet R0/12 : sensor
|   |
|   +-7014 (cevSensorModuleDeviceTemp) : Temp: Asic1 R0/13 : sensor
|   |
|   +-7026 (cevModuleCpuType) : cpu R0/0 : other
|   |
|   +-7027 (cevPortUSB) : usb R0/0 : port
|   |
|   \-7029 (cevPortGe) : NME R0 : port

```

Mib Variables printed : <entPhysicalName entPhysicalClass>

Generating SNMP Notifications

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

- [Identifying Hosts to Receive Notifications](#)
- [Configuration Changes](#)
- [FRU Status Changes](#)

Identifying Hosts to Receive Notifications

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 (notifications 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: `notification(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 notifications. For example, to generate `mplsLdpSessionUp` or `mplsLdpSessionDown` notifications, the MPLS-LDP-MIB object `mplsLdpSessionUpDownTrapEnable` must be set to `enabled(1)`.

Configuration Changes

If entity notifications are enabled, the router generates an `entConfigChange` notification (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 checks the value of the `entLastChangeTime` object to detect any `entConfigChange` notifications that were missed as a result of throttling or transmission loss.

Enabling notifications for Configuration Changes

To configure the router to generate an `entConfigChange` notification each time its configuration changes, enter the following command from the CLI. Use the **no** form of the command to disable the notifications.

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

FRU Status Changes

If FRU notifications are enabled, the router generates the following notifications in response to changes in the status of an FRU:

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



Note See the `CISCO-ENTITY-FRU-CONTROL-MIB` for more information about these notifications.

Enabling FRU Notifications

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

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

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

Monitoring Quality of Service

This section provides the following information about using Quality of Service (QoS) in your configuration:

- `CISCO-CLASS-BASED-QOS-MIB` Overview
- Viewing QoS Configuration Settings Using the `CISCO-CLASS-BASED-QOS-MIB`
- Monitoring QoS Using the `CISCO-CLASS-BASED-QOS-MIB`
- Considerations for Processing QoS Statistics
- Sample QoS Applications

CISCO-CLASS-BASED-QOS-MIB Overview

The `CISCO-CLASS-BASED-QOS-MIB` provides read only access to quality of service (QoS) configuration information and statistics for Cisco platforms that support the modular Quality of Service command-line interface (modular QoS CLI).

CISCO-CLASS-BASED-QOS-MIB Object Relationship

To understand how to navigate the `CISCO-CLASS-BASED-QOS-MIB` tables, it is important to understand the relationship among different QoS objects. QoS objects consists of:

- Match Statement—specific match criteria to identify packets for classification purposes.
- Class Map—a user-defined traffic class that contains 1 or more match statements used to classify packets into different categories.
- Feature Action—a QoS feature. Features include police, traffic shaping, queueing, random detect, and packet marking. After the traffic has been classified we apply actions to each traffic class.
- Policy Map—a user-defined policy that associates a QoS feature action to the user-define class map.

- Service Policy—a policy map that has been attached to an interface.

The MIB uses the following indices to identify QoS features and distinguish among instances of those features:

- cbQosObjectsIndex – identifies each QoS feature on the router.
- cbQosConfigIndex n- identifies a type of QoS configuration. This index is shared by QoS objects that have identical configuration.
- cbQosPolicyIndex – identifies a unique service policy.

QoS MIB Information Storage

CISCO-CLASS-BASED-QOS-MIB information is stored in:

- Configuration instances – includes all class maps, policy map, match statements, and feature action configuration parameters. Might have multiple identical instances. Multiple instances of the same QoS feature share a single configuration object, which is identified by cbQosConfigIndex.
- Runtime Statistics instances—Includes summary counts and rates by traffic class before and after any configured QoS policies are enforced. In addition, detailed feature-specific statistics are available for select PolicyMap features. Each has a unique runtime instance. Multiple instances of a QoS feature have a separate statistics object. Run-time instances of QoS objects are each assigned a unique identifier (cbQosObjectsIndex) to distinguish among multiple objects with matching configurations.

Viewing QoS Configuration Settings Using the CISCO-CLASS-BASED-QOS-MIB

This section contains examples that show how QoS configuration settings are stored in CISCO-CLASS-BASED-QOS-MIB tables. The samples show information grouped by QoS object; however, the actual output of an SNMP query might show QoS information similar to the following.



Note

This is only a partial display of all QoS information.

```
getmany -v2c 9.0.0.55 ciscoCBQosMIB
cbQosIfType.64 = mainInterface(1)
cbQosIfType.66 = mainInterface(1)
cbQosPolicyDirection.64 = input(1)
cbQosPolicyDirection.66 = output(2)
cbQosIfIndex.64 = 4
cbQosIfIndex.66 = 4
cbQosFrDLCI.64 = 0
cbQosFrDLCI.66 = 0
cbQosAtmVPI.64 = 0
cbQosAtmVPI.66 = 0
cbQosAtmVCI.64 = 0
cbQosAtmVCI.66 = 0
cbQosEntityIndex.64 = 0
cbQosEntityIndex.66 = 0
cbQosConfigIndex.64.64 = 15348192
cbQosConfigIndex.64.7282691 = 12103539
cbQosConfigIndex.64.15123441 = 1593
cbQosConfigIndex.64.15755442 = 1594
cbQosConfigIndex.66.66 = 15889568
cbQosConfigIndex.66.1907619 = 15971699
cbQosConfigIndex.66.9319458 = 1594
```

```

cbQosConfigIndex.66.15082481 = 1593
cbQosObjectsType.64.64 = policymap(1)
cbQosObjectsType.64.7282691 = police(7)
cbQosObjectsType.64.15123441 = classmap(2)
cbQosObjectsType.64.15755442 = matchStatement(3)
cbQosObjectsType.66.66 = policymap(1)
cbQosObjectsType.66.1907619 = queueing(4)
cbQosObjectsType.66.9319458 = matchStatement(3)
cbQosObjectsType.66.15082481 = classmap(2)
cbQosParentObjectsIndex.64.64 = 0
cbQosParentObjectsIndex.64.7282691 = 15123441
cbQosParentObjectsIndex.64.15123441 = 64
cbQosParentObjectsIndex.64.15755442 = 15123441
cbQosParentObjectsIndex.66.66 = 0
cbQosParentObjectsIndex.66.1907619 = 15082481
cbQosParentObjectsIndex.66.9319458 = 15082481
cbQosParentObjectsIndex.66.15082481 = 66
cbQosPolicyMapName.15348192 = policy-police
cbQosPolicyMapName.15889568 = policy-bw
cbQosPolicyMapDesc.15348192 =
cbQosPolicyMapDesc.15889568 =
cbQosCMName.1593 = class-default
cbQosCMDesc.1593 =
cbQosCMInfo.1593 = matchAny(3)
.....
.....

```

Monitoring QoS Using the CISCO-CLASS-BASED-QOS-MIB

This section describes how to monitor QoS on the router by checking the QoS statistics in the CISCO-CLASS-BASED-QOS-MIB tables.



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 lists the types of QoS statistics tables.

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.
cbQosMatchStmntStatsTable	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.

Table A-1 QoS Statistics Tables

QoS Table	Statistics
cbQoSStatsTable	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 samples, these counters are shown as 33-bit counters.

When accessing QoS counter statistics, 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.

Sample QoS Statistics Tables

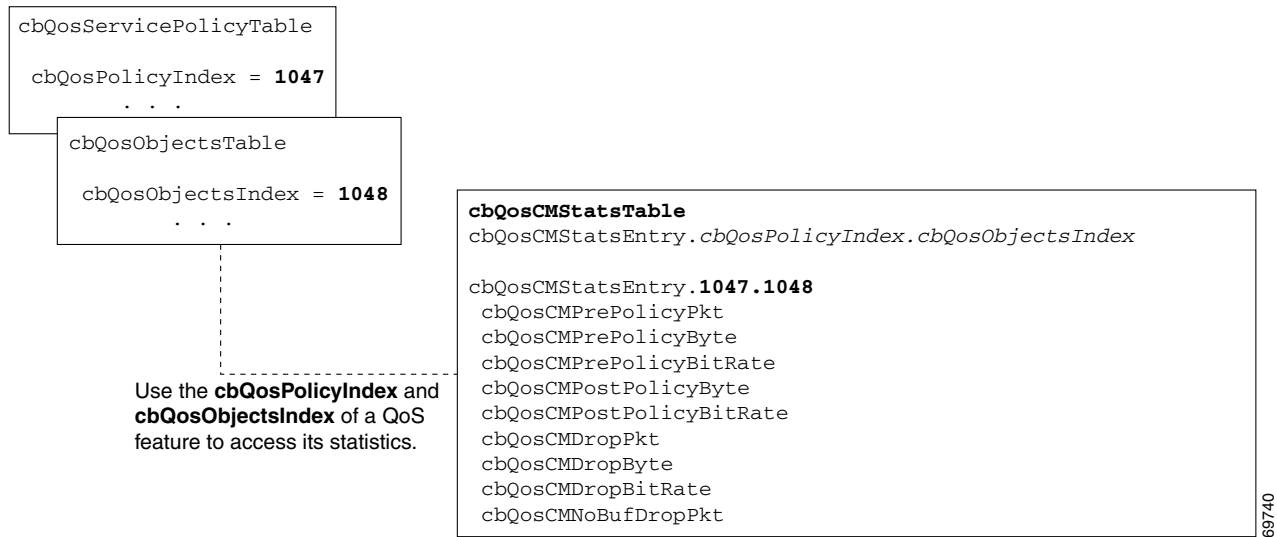
The samples in this section show the counters in CISCO-CLASS-BASED-QOS-MIB statistics tables:

- [Figure A-2](#) shows the counters in the *cbQosCMStatsTable* and the indexes for accessing these and other statistics.
- [Figure A-3](#) shows the counters in *cbQosMatchStmntStatsTable*, *cbQosPoliceStatsTable*, *cbQosQueueingStatsTable*, *cbQosTSSStatsTable*, and *cbQosREDClassStatsTable*.

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*

Figure A-2 QoS Class Map Statistics and Indexes



- [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 if 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 if a service policy has been associated with the transmit and receive directions of a customer interface. Mark noncustomer 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-IF
END-WHILE
RETURN total;

```

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 Notifications

To determine if an interface has failed, you can monitor `linkDown` and `linkUp` notifications for the interface. See the [“Enabling Interface linkUp/linkDown Notifications”](#) section on page A-33 for instructions on how to enable these notifications.

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

Enabling Interface linkUp/linkDown Notifications

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` notifications:

-
- Step 1** Issue the following CLI command to enable `linkUp` and `linkDown` notifications 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` notifications are enabled or disabled for that interface.
- Step 3** To enable `linkUp` and `linkDown` notifications on an interface, set `ifLinkUpDownTrapEnable` to `enabled(1)`. To configure the router to send `linkDown` notifications only for the lowest layer of an interface, see the [“SNMP Notification Filtering for linkDown Notifications”](#) section on page A-34.

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

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

- Step 5** To disable notifications, use the **no** form of the appropriate command.
- 

## SNMP Notification Filtering for linkDown Notifications

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

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

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

## Billing Customers for Traffic

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

This section contains the following topics:

- [Input and Output Interface Counts, page A-34](#)
- [Determining the Amount of Traffic to Bill to a Customer, page A-35](#)
- [Scenario for Demonstrating QoS Traffic Policing, page A-35](#)

## 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.

For detailed constraints about IF-MIB counter support, see the [“IF-MIB \(RFC 2863\)” section on page 3-97](#).

Read the following important information about the IF-MIB counter support:

- Unless noted, all IF-MIB counters are supported on Cisco ASR 1000 Series Routers interfaces.
- For IF-MIB high capacity counter support, Cisco conforms to the RFC 2863 standard. The RFC 2863 standard states that for interfaces that operate:
  - At 20 million bits per second or less, 32-bit byte and packet counters *must* be supported.
  - Faster than 20 million bits per second and slower than 650,000,000 bits per second, 32-bit packet counters and 64-bit octet counters *must* be supported.
  - At 650,000,000 bits per second or faster, 64-bit packet counters *and* 64-bit octet counters *must* be supported.

- 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 need this information in the following steps.
  - Step 3** Generate traffic with the traffic generator. The data rate should be more than that is configured for Conform burst(bc)/Exceed burst(be) for the policy.
  - 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 is 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 ASR 1000 Series Routers 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 ASR 1000 Series Router Software Configuration Guide*.

```

Policy Map test-police
 Class class-default
 police cir 1000000 bc 10000 be 20000
 conform-action transmit
 exceed-action drop
 violate-action drop

interface GigabitEthernet1/1/5
 ip address 15.1.0.52 255.0.0.0
 no negotiation auto
 service-policy output test-police
end

```

## 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

```

Router#sh policy-map interface gi 1/1/5

GigabitEthernet1/1/5

Service-policy output: test-police

Class-map: class-default (match-any)
 0 packets, 0 bytes
 5 minute offered rate 0 bps, drop rate 0 bps
 Match: any
 police:
 cir 1000000 bps, bc 10000 bytes, be 20000 bytes
 conformed 0 packets, 0 bytes; actions:
 transmit
 exceeded 0 packets, 0 bytes; actions:
 drop
 violated 0 packets, 0 bytes; actions:
 drop
 conformed 0 bps, exceed 0 bps, violate 0 bps

```

### SNMP Output

```

ptolemy:4> getmany 9.0.0.52 cbQosIfIndex
cbQosIfIndex.290 = 18
ptolemy:5> getone 9.0.0.52 ifDescr.18
ifDescr.18 = GigabitEthernet1/1/5
ptolemy:6>

getmany 9.0.0.52 cbQosCMDropPkt cbQosCMDropByte
cbQosCMDropPkt.290.9756705 = 0
cbQosCMDropByte.290.9756705 = 0
ptolemy:77>

```

## Packet Counts After the Service Policy Is Applied

After you generate traffic using the traffic generator, look at the number of packets that exceeded and conformed to the committed information rate (CIR) set by the `police` command:

- 19351 packets conformed to the police rate and were transmitted
- 80 packets exceeded the police rate and were dropped
- 16066130 packets violated the police rate and were dropped

The following CLI and SNMP output show the counts on the interface after the service policy is applied. The object `cbQosCMDropPkt` refers to sum of exceeded and violated packets and `cbQosCMDropByte` refers to the sum of exceeded and violated bytes. (In the output, exceeded and violated packet counts are shown in boldface.)

### CLI Command Output

```
Router#sh show policy-map int gi 1/1/5

GigabitEthernet1/1/5

Service-policy output: test-police

Class-map: class-default (match-any)
 16085561 packets, 1994609369 bytes

 5 minute offered rate 16051000 bps, drop rate 16032000 bps
Match: any
police:
 cir 1000000 bps, bc 10000 bytes, be 10000 bytes
 conformed 19351 packets, 2399329 bytes; actions:
 transmit
 exceeded 80 packets, 9920 bytes; actions:
 drop
 violated 16066130 packets, 1992200120 bytes; actions:
 drop
 conformed 0 bps, exceed 0 bps, violate 16032000 bps

Router#
```

### SNMP Output

```
getmany 9.0.0.52 cbQosCMDropPkt cbQosCMDropByte
cbQosCMDropPkt.290.9756705 = 16066210
cbQosCMDropByte.290.9756705 = 1992210040
ptolemy:77>
. . .
```

## Using IF-MIB Counters

This section describes the IF-MIB counters and how you can use them on various interfaces and subinterfaces. The subinterface counters are specific to the protocols. This section addresses the IF-MIB counters for ATM interfaces.

The IF-MIB counters are defined with respect to lower and upper layers:

- `ifInDiscards`—The number of inbound packets which were discarded, even though no errors were detected to prevent their being deliverable to a higher-layer protocol. One reason for discarding such a packet could be to free up buffer space.

- **IfInErrors**—The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol for packet-oriented interfaces.
- **ifInUnknownProtos**—The number of packets received through the interface which were discarded because of an unknown or unsupported protocol for packet-oriented interfaces.
- **ifOutDiscards**—The number of outbound packets which were discarded even though no errors were detected to prevent their being transmitted. One reason for discarding such a packet is to free up buffer space.
- **ifOutErrors**—The number of outbound packets that could not be transmitted because of errors for packet-oriented interfaces.

The logical flow for counters works as follows:

1. When a packet arrives on an interface, check for the following:
  - a. Error in packet—If any errors are detected, increment **ifInErrors** and drop the packet.
  - b. Protocol errors—If any errors are detected, increment **ifInUnknownProtos** and drop the packet.
  - c. Resources (buffers)—If unable to get resources, increment **ifInDiscards** and drop the packet.
  - d. Increment **ifInUcastPkts/ ifInNUcastPkts** and process the packet (At this point, increment the **ifInOctets** with the size of packet).
2. When a packet is to be sent out of an interface:
  - a. Increment **ifOutUcastPkts/ ifOutNUcastPkts** (Here we also increment **ifOutOctets** with the size of packet).
  - b. Check for error in packet and if there are any errors in packet, increment **ifOutErrors** and drop the packet.
  - c. Check for resources (buffers) and if you cannot get resources then increment **ifOutDiscards** and drop packet.

This following output is an example IF-MIB entries:

IfXEntry ::=

```
SEQUENCE {
 ifName DisplayString,
 ifInMulticastPkts Counter32,
 ifInBroadcastPkts Counter32,
 ifOutMulticastPkts Counter32,
 ifOutBroadcastPkts Counter32,
 ifHCInOctets Counter64,
 ifHCInUcastPkts Counter64,
 ifHCInMulticastPkts Counter64,
 ifHCInBroadcastPkts Counter64,
 ifHCOutOctets Counter64,
 ifHCOutUcastPkts Counter64,
 ifHCOutMulticastPkts Counter64,
 ifHCOutBroadcastPkts Counter64,
 ifLinkUpDownTrapEnable INTEGER,
 ifHighSpeed Gauge32,
 ifPromiscuousMode TruthValue,
 ifConnectorPresent TruthValue,
 ifAlias DisplayString,
 ifCounterDiscontinuityTime TimeStamp
}
```

## Sample Counters

The high capacity counters are 64-bit versions of the basic ifTable counters. They have the same basic semantics as their 32-bit counterparts; their syntax is extended to 64 bits.

Table A-2 lists capacity counter object identifiers (OIDs).

**Table A-2 Capacity Counters Object Identifiers**

| Name                       | Object Identifier (OID) |
|----------------------------|-------------------------|
| ifHCInOctets               | ::= { ifXEntry 6 }      |
| ifHCInUcastPkts            | ::= { ifXEntry 7 }      |
| ifHCInMulticastPkts        | ::= { ifXEntry 8 }      |
| ifHCInBroadcastPkts        | ::= { ifXEntry 9 }      |
| ifHCOctets                 | ::= { ifXEntry 10 }     |
| ifHCOUcastPkts             | ::= { ifXEntry 11 }     |
| ifHCOUMulticastPkts        | ::= { ifXEntry 12 }     |
| ifHCOUBroadcastPkts        | ::= { ifXEntry 13 }     |
| ifLinkUpDownTrapEnable     | ::= { ifXEntry 14 }     |
| ifHighSpeed                | ::= { ifXEntry 15 }     |
| ifPromiscuousMode          | ::= { ifXEntry 16 }     |
| ifConnectorPresent         | ::= { ifXEntry 17 }     |
| ifAlias                    | ::= { ifXEntry 18 }     |
| ifCounterDiscontinuityTime | ::= { ifXEntry 19 }     |

## Related Information and Useful Links

The following URLs provide access to helpful information about Cisco IF-MIB counters:

- Frequently asked questions about SNMP counters:  
[http://www.cisco.com/en/US/customer/tech/tk648/tk362/technologies\\_q\\_and\\_a\\_item09186a00800b69ac.shtml](http://www.cisco.com/en/US/customer/tech/tk648/tk362/technologies_q_and_a_item09186a00800b69ac.shtml)
- Access Cisco IOS MIB Tools from the following URL:  
<http://tools.cisco.com/ITDIT/MIBS/servlet/index>

## Overview of SIPs and SPAs

The following list describes some of the general characteristics of Cisco SIPs and SPAs (shared port adapter).

- A Cisco ASR 1000 Series SPA Interface Processor (SIP) is a carrier card that:
  - Inserts into a router slot like a line card. It provides no network connectivity on its own.
  - Contains one or more subslots, which are used to house one or more SPAs. The SPA provides interface ports for network connectivity.

- Resides in the router fully populated either with functional SPAs in all subslots during normal operation or with a blank filler plate (SPA-BLANK=) inserted in all empty subslots.
- Support online insertion and removal (OIR) with SPAs inserted in their subslots. SPAs also support OIR and can be inserted or removed independently from the SIP.
- A Shared Port Adapter (SPA) is a modular type of port adapter that:
  - Inserts into a subslot of a compatible SIP carrier card to provide network connectivity and increased interface port density. A SIP can hold one or more SPAs, depending on the SIP type.
  - Provides services rather than network connectivity and insert into subslots of compatible cards. For example, the IPSec VPN SPA provides services such as IP Security (IPSec) encryption/decryption, generic routing encapsulation (GRE ), and Internet Key Exchange (IKE) key generation.
  - Are available in single-height (inserts into one SIP subslot) and double-height (inserts into two single, vertically aligned SIP subslots).

**Note**


---

SPA-1X10GE-WL-V2 is supported on the Cisco ASR1K platform beginning with Cisco IOS XE Release 3.3.0 S and Cisco IOS Release 15.1(2)S.

---

**Note**


---

The 1-Port 10GE LAN/WAN-PHY Shared Port Adapter (SPA-1X10GE-WL-V2) should be on the same mode, either the LAN mode or the WAN mode, at both ends.

---

**Note**


---

The SPA-1X10GE-WL-V2 (configured in the LAN mode) is compatible with the SPA-1X10GE-L-V2 (LAN SPA).

---

## Configuring the LAN-PHY Mode

Use the following commands to configure the LAN-PHY mode on the 1-Port 10GE LAN/WAN-PHY Shared Port Adapter (SPA-1X10GE-WL-V2):

```
show controllers wanphy interface-path-id [alarms | all | registers]
configure terminal
controller wanphy interface-path-id
lanmode on
end
hw-module subslot interface-path-id reload
show controllers wanphy interface-path-id [alarms | all | registers]
```

**Note**


---

After configuring the LAN-PHY mode and reloading the SPA, all the links are in the UP state.

---

**Note**


---

Effective from Cisco IOS Release 15.1(2)S, 1-Port 10GE LAN/WAN-PHY Shared Port Adapter (SPA-1X10GE-WL-V2) supports both the LAN and WAN modes.

---

## Displaying the SIP Hardware Type

To verify the SIP hardware type that is installed in your Cisco ASR 1000 series router, you can use the `show platform` command. There are some commands on the Cisco ASR 1000 series router that provide SIP hardware information. There are more sub-commands which give detailed output for each SIP/SPA card. The example below shows some list of such commands.

### Example A-9 Example of the `show platform` command

The following example shows the output of the `show platform` command on the Cisco ASR 1000 Series Routers:

```
Router#sh platform
```

```
Chassis type: ASR1006
```

| Slot | Type            | State      | Insert time (ago) |
|------|-----------------|------------|-------------------|
| 0    | ASR1000-SIP10   | ok         | 06:19:03          |
| 0/0  | SPA-1XOC12-POS  | ok         | 06:17:25          |
| 0/1  | SPA-2XCT3/DS0   | ok         | 06:17:25          |
| 0/2  | SPA-2XT3/E3     | ok         | 06:17:25          |
| 0/3  | SPA-8X1GE-V2    | ok         | 06:17:34          |
| 1    | ASR1000-SIP10   | ok         | 06:19:03          |
| 1/0  | SPA-1X10GE-L-V2 | ok         | 06:17:36          |
| 1/1  | SPA-5X1GE-V2    | ok         | 06:17:25          |
| 1/2  | SPA-8X1FE-TX-V2 | ok         | 06:17:36          |
| 2    | ASR1000-SIP10   | ok         | 06:19:03          |
| 2/0  | SPA-2X1GE-V2    | ok         | 06:17:36          |
| 2/1  | SPA-10X1GE-V2   | ok         | 06:17:36          |
| 2/2  | SPA-2XOC3-POS   | ok         | 06:17:36          |
| R0   | ASR1000-RP1     | ok, active | 06:19:03          |
| R1   |                 | unknown    | 06:19:03          |
| F0   | ASR1000-ESP10   | ok, active | 06:19:03          |
| P0   | ASR1006-PWR-AC  | ok         | 06:18:25          |
| P1   | ASR1006-FAN     | ok         | 06:18:25          |

| Slot | CPLD Version | Firmware Version                        |
|------|--------------|-----------------------------------------|
| 0    | 06120701     | 12.2(20070802:195019) [gschnorr-mcp_... |
| 1    | 06120701     | 12.2(20070802:195019) [gschnorr-mcp_... |
| 2    | 06120701     | 12.2(20070802:195019) [gschnorr-mcp_... |

```

R0 0706210B 12.2(20070807:170946) [gschnorr-mcp_...
R1 N/A N/A
F0 07021400 12.2(20070802:195019) [gschnorr-mcp_...

```

```

Router#sh platform ?
 hardware Show platform hardware information
 software Show platform software information
 | Output modifiers
<cr>

```

```

Router#sh platform har
Router#sh platform hardware ?
 cpp Cisco packet processor
 interface Interface information
 port port information
 slot Slot information
 subslot Subslot information

```

```

Router#sh platform hardware slot ?
 0 SPA-Inter-Processor slot 0
 1 SPA-Inter-Processor slot 1
 2 SPA-Inter-Processor slot 2
 F0 Embedded-Service-Processor slot 0
 F1 Embedded-Service-Processor slot 1
 R0 Route-Processor slot 0
 R1 Route-Processor slot 1

```

```

Router#sh platform hardware slot 0 ?
 dram MCP85xx DRAM commands
 eobc Show EOBC
 fan Fan commands
 io-port IO Port information
 led LED-related commands
 mcu MCU related commands
 plim PLIM information
 sensor Sensor information

```

```
serdes Serdes information
spa SPA related information
voltage Voltage commands
```

```
Router#
```

