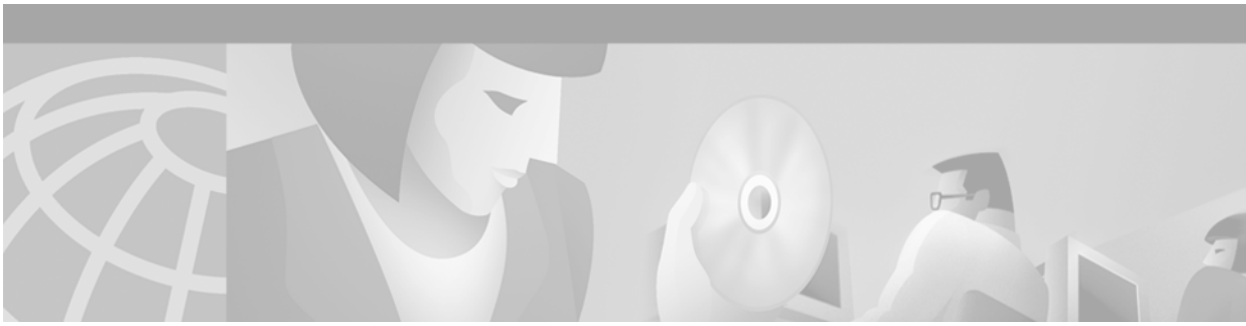

Configuring the Modular Quality of Service Command-Line Interface

This section contains the tasks for configuring QoS functionality using the Modular Quality of Service (QoS) Command-Line Interface (CLI) (MQC).

For complete conceptual information, see the chapter "Modular Quality of Service Command-Line Interface Overview" in this book.

For a complete description of the QoS commands in this chapter, refer to the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T. To locate documentation of other commands that appear in this chapter, use the command reference master index or search online.

To identify the hardware platform or software image information associated with a feature, use the Feature Navigator on Cisco.com to search for information about the feature or refer to the software release notes for a specific release. For more information, see the "Identifying Supported Platforms" section in the "Using Cisco IOS Software" chapter in this book.

Modular QoS CLI Configuration Task List

To configure and enable class-based QoS features, perform the tasks described in the following sections. The tasks in the first three sections are required; the task in the fourth section is optional.

- Creating a Traffic Class (Required)
- Creating a Traffic Policy (Required)
- Attaching a Traffic Policy to an Interface (Required)
- Verifying the Traffic Class and Traffic Policy Information (Optional)

See the end of this chapter for the section "Modular QoS CLI Configuration Examples."

Creating a Traffic Class

To create a traffic class, use the **class-map** command. The syntax of the **class-map** command is as follows:

class-map [**match-any** | **match-all**] *class-name*
no class-map [**match-any** | **match-all**] *class-name*

The match-all and match-any Keywords

The **match-all** and **match-any** keywords need to be specified only if more than one match criterion is configured in the traffic class.

The **match-all** keyword is used when *all* of the match criteria in the traffic class must be met in order for a packet to be placed in the specified traffic class.

The **match-any** keyword is used when only *one* of the match criterion in the traffic class must be met in order for a packet to be placed in the specified traffic class.

If neither the **match-all** nor **match-any** keyword is specified, the traffic class will behave in a manner consistent with **match-all** keyword.

About The match not Command

The **match not** command, rather than identifying the specific match parameter to use as a match criterion, is used to specify a match criterion that prevents a packet from being classified as a member of the class. For instance, if the **match not qos-group 6** command is issued while you configure the traffic class, QoS group 6 becomes the only QoS group value that is not considered a successful match criterion. All other QoS group values would be successful match criteria.

Procedure

To create a traffic class containing match criteria, use the **class-map** command to specify the traffic class name. Then use one or more **match** commands to specify the appropriate match criteria. Packets matching the criteria you specify are placed in the traffic class.

Note In the following steps, a number of **match** commands are listed. The specific **match** commands available vary by platform and Cisco IOS release. For the **match** commands available, see the Cisco IOS command reference for the platform and Cisco IOS release you are using.

	Command or Action	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# class-map [match-all | match-any] class-name`	Creates a class to be used with a class map, and enters class-map configuration mode. The class map is used for matching packets to the specified class. **Note** The **match-all** keyword specifies that all match criteria must be met. The **match-any** keyword specifies that one of the match criterion must be met.
	Use one or more of the following **match** commands, as applicable.	
Step 4	`Router(config-cmap)# match access-group {access-group | name access-group-name}`	(Optional) Configures the match criteria for a class map on the basis of the specified access control list (ACL). **Note** Access lists configured with the optional **log** keyword of the **access-list** command are not supported when configuring a traffic class. For more information about the **access-list** command, see the *Cisco IOS IP Application Services Command Reference,* Release 12.4 T.

	Command or Action	Purpose
Step 5	`Router(config-cmap)# match any`	(Optional) Configures the match criteria for a class map to be successful match criteria for all packets.
Step 6	`Router config-cmap)# match class-map class-name`	(Optional) Specifies the name of a traffic class to be used as a matching criterion (for nesting traffic class [nested class maps] within one another).
Step 7	`Router(config-cmap)# match cos cos-number`	(Optional) Matches a packet based on a Layer 2 class of service (CoS) marking.
Step 8	`Router(config-cmap)# match destination-address mac address`	(Optional) Uses the destination Media Access Control (MAC) address as a match criterion.
Step 9	`Router(config-cmap)# match discard-class class-number`	(Optional) Matches packets of a certain discard class.
Step 10	`Router(config-cmap)# match [ip] dscp dscp-value [dscp-value dscp-value dscp-value dscp-value dscp-value dscp-value]`	(Optional) Identifies a specific IP differentiated service code point (DSCP) value as a match criterion. Up to eight DSCP values can be included in one match statement.
Step 11	`Router(config-cmap)# match field protocol protocol-field {eq [mask] | neq [mask] | gt | lt | range range | regex string} value [next next-protocol]`	(Optional) Configures the match criteria for a class map on the basis of the fields defined in the protocol header description files (PHDFs).
Step 12	`Router(config-cmap)# match fr-dlci dlci-number`	(Optional) Specifies the Frame Relay data-link connection identifier (DLCI) number as a match criterion in a class map.
Step 13	`Router(config-cmap)# match input-interface interface-name`	(Optional) Configures a class map to use the specified input interface as a match criterion.
Step 14	`Router(config-cmap)# match ip rtp starting-port-number port-range`	(Optional) Configures a class map to use the Real-Time Protocol (RTP) protocol port as the match criterion.
Step 15	`Router(config-cmap)# match mpls experimental mpls-values`	(Optional) Configure a class map to use the specified value of the Multiprotocol Label Switching (MPLS) experimental (EXP) field as a match criterion.
Step 16	`Router(config-cmap)# match mpls experimental topmost values`	(Optional) Matches the MPLS EXP value in the topmost label.
Step 17	`Router(config-cmap)# match not match-criteria`	(Optional) Specifies the single match criterion value to use as an unsuccessful match criterion.
Step 18	`Router(config-cmap)# match packet length {max maximum-length-value [min minimum-length-value] | min minimum-length-value [max maximum-length-value]}`	(Optional) Specifies the Layer 3 packet length in the IP header as a match criterion in a class map.
Step 19	`Router(config-cmap)# match port-type {routed | switched}`	(Optional) Matches traffic on the basis of the port type for a class map.
Step 20	`Router(config-cmap)# match [ip] precedence precedence-value [precedence-value precedence-value precedence-value]`	(Optional) Identifies IP precedence values as match criteria.

	Command or Action	Purpose
Step 21	`Router(config-cmap)# match protocol protocol-name`	(Optional) Configures the match criteria for a class map on the basis of the specified protocol. **Note** There is a separate **match protocol** (NBAR) command used to configure network-based application recognition (NBAR) to match traffic by a protocol type known to NBAR.
Step 22	`Router(config-cmap)# match protocol citrix [app application-name-string] [ica-tag ica-tag-value]`	(Optional) Configures NBAR to match Citrix traffic.
Step 23	`Router(config-cmap)# match protocol fasttrack file-transfer "regular-expression"`	(Optional) Configures NBAR to match FastTrack peer-to-peer traffic.
Step 24	`Router(config-cmap)# match protocol gnutella file-transfer "regular-expression"`	(Optional) Configures NBAR to match Gnutella peer-to-peer traffic.
Step 25	`Router(config-cmap)# match protocol http [url url-string | host hostname-string | mime MIME-type | c-header-field c-header-field-string | s-header-field s-header-field-string]`	(Optional) Configures NBAR to match Hypertext Transfer Protocol (HTTP) traffic by URL, host, Multipurpose Internet Mail Extension (MIME) type, or fields in HTTP packet headers.
Step 26	`Router(config-cmap)# match protocol rtp [audio | video | payload-type payload-string]`	(Optional) Configures NBAR to match Real-Time Transfer Protocol (RTP) traffic.
Step 27	`Router(config-cmap)# match qos-group qos-group-value`	(Optional) Identifies a specific QoS group value as a match criterion.
Step 28	`Router(config-cmap)# match source-address mac address-destination`	(Optional) Uses the source MAC address as a match criterion.
Step 29	`Router(config-cmap)# match start {l2-start | l3-start} offset number size number {eq | neq | gt | lt | range range | regex string} {value [value2] | [string]}`	(Optional) Configures the match criteria for a class map on the basis of the datagram header (Layer 2) or the network header (Layer 3).
Step 30	`Router(config-cmap)# match tag {tag-name}`	(Optional) Specifies tag type as a match criterion.
Step 31	`Route(config-cmap)# exit`	(Optional) Exits class-map configuration mode.

Creating a Traffic Policy

To configure a traffic policy (sometimes also referred to as a policy map), use the **policy-map** command. The **policy-map** command allows you to specify the traffic policy name and also allows you to enter policy-map configuration mode (a prerequisite for enabling QoS features such as traffic policing or traffic shaping).

Associate the Traffic Policy with the Traffic Class

After using the **policy-map** command, use the **class** command to associate the traffic class (created in the "Creating a Traffic Class" section on page 405) with the traffic policy.

The syntax of the **class** command is as follows:

class *class-name*
no class *class-name*

For the *class-name* argument, use the name of the class you created when you used the **class-map** command to create the traffic class (Step 3 of the "Creating a Traffic Class" section on page 405).

After entering the **class** command, you are automatically in policy-map class configuration mode. The policy-map class configuration mode is the mode used for enabling the specific QoS features.

Procedure

To create a traffic policy (or policy map) and enable one or more QoS features, perform the following steps.

Note This procedure lists many of the commands you can use to enable one or more QoS features. For example, to enable Class-Based Weighted Fair Queuing (CBWFQ), you would use the **bandwidth** command. Not all QoS features are available on all platforms or in all Cisco IOS releases. For the features and commands available to you, see the Cisco IOS documentation for your platform and version of Cisco IOS software you are using.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# policy-map policy-name`	Creates or specifies the name of the traffic policy and enters policy-map configuration mode.
Step 4	`Router(config-pmap)# class {class-name | class-default}`	Specifies the name of a traffic class (previously created in the "Creating a Traffic Class" section on page 405) and enters policy-map class configuration mode.
	Use one or more of the following commands to enable the specific QoS feature you want to use.	
Step 5	`Router(config-pmap-c)# bandwidth {bandwidth-kbps | percent percent}`	(Optional) Specifies a minimum bandwidth guarantee to a traffic class in periods of congestion. A minimum bandwidth guarantee can be specified in kbps or by a percentage of the overall available bandwidth.
Step 6	`Router(config-pmap-c)# fair-queue number-of-queues`	(Optional) Specifies the number of queues to be reserved for a traffic class.
Step 7	`Router (config-pmap-c)# police bps [burst-normal] [burst-max] conform-action action exceed-action action [violate-action action]`	(Optional) Configures traffic policing.
Step 8	`Router(config-pmap-c)# priority {bandwidth-kbps | percent percentage} [burst]`	(Optional) Gives priority to a class of traffic belonging to a policy map.
Step 9	`Router(config-pmap-c)# queue-limit number-of-packets`	(Optional) Specifies or modifies the maximum number of packets the queue can hold for a class configured in a policy map.
Step 10	`Router(config-pmap-c)# random-detect [dscp-based | prec-based]`	(Optional) Enables Weighted Random Early Detection (WRED) or distributed WRED (DWRED).
Step 11	`Router(config-pmap-c)# set atm-clp`	(Optional) Sets the cell loss priority (CLP) bit when a policy map is configured.

	Command	Purpose
Step 12	`Router(config-pmap-c)# set cos {cos-value | from-field [table table-map-name]}`	(Optional) Sets the Layer 2 class of service (CoS) value of an outgoing packet.
Step 13	`Router(config-pmap-c)# set discard-class value`	(Optional) Marks a packet with a discard-class value.
Step 14	`Router(config-pmap-c)# set [ip] dscp {dscp-value | from-field [table table-map-name]}`	(Optional) Marks a packet by setting the differentiated services code point (DSCP) value in the type of service (ToS) byte.
Step 15	`Router(config-pmap-c)# set fr-de`	(Optional) Changes the discard eligible (DE) bit setting in the address field of a Frame Relay frame to 1 for all traffic leaving an interface.
Step 16	`Router(config-pmap-c)# set precedence {precedence-value | from-field [table table-map-name]}`	(Optional) Sets the precedence value in the packet header.
Step 17	`Route(config-pmap-c)# set mpls experimental value`	(Optional) Designates the value to which the MPLS bits are set if the packets match the specified policy map.
Step 18	`Router (config-pmap-c)# set qos-group {group-id | from-field [table table-map-name]}`	(Optional) Sets a QoS group identifier (ID) that can be used later to classify packets.
Step 19	`Router(config-pmap-c)# service-policy policy-map-name`	(Optional) Specifies the name of a traffic policy used as a matching criterion (for nesting traffic policies [hierarchical traffic policies] within one another).
Step 20	`Router(config-pmap-c)# shape {average | peak} mean-rate [burst-size [excess-burst-size]]`	(Optional) Shapes traffic to the indicated bit rate according to the algorithm specified.
Step 21	`Router(config-pmap-c)# exit`	(Optional) Exits policy-map class configuration mode.

Attaching a Traffic Policy to an Interface

To attach a traffic policy to an interface, use the **service-policy** command. The **service-policy** command also allows you to specify the direction in which the traffic policy should be applied (either on packets coming into the interface or packets leaving the interface).

The **service-policy** command syntax is as follows:

service-policy {**input** | **output**} *policy-map-name*
no service-policy {**input** | **output**} *policy-map-name*

Procedure

To attach a traffic policy to an interface, perform the following steps.

Note Depending on the platform and Cisco IOS release you are using, a traffic policy can be attached to an ATM permanent virtual circuit (PVC) subinterface, a Frame Relay data-link connection identifier (DLCI), or another type of interface.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# interface serial0`	Configures an interface type and enters interface configuration mode.
Step 4	`Router(config-if)# service-policy output [type access-control] {input | output} policy-map-name`	Attaches a policy map to an interface.
Step 5	`Router (config-if)# exit`	(Optional) Exits interface configuration mode.

Note Multiple traffic policies on tunnel interfaces and physical interfaces are not supported if the interfaces are associated with each other. For instance, if a traffic policy is attached to a tunnel interface while another traffic policy is attached to a physical interface with which the tunnel interface is associated, only the traffic policy on the tunnel interface works properly.

Verifying the Traffic Class and Traffic Policy Information

To display and verify the information about a traffic class or traffic policy, perform the following steps.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# show class-map [type {stack | access-control}] [class-map-name]`	(Optional) Displays all class maps and their matching criteria.
Step 3	`Router# show policy-map policy-map class class-name`	(Optional) Displays the configuration for the specified class of the specified policy map.
Step 4	`Router# show policy-map policy-map`	(Optional) Displays the configuration of all classes for a specified policy map or all classes for all existing policy maps.
Step 5	`Router# show policy-map interface [type access-control] type number [vc [vpi/] vci] [dlci dlci] [input | output]`	(Optional) Displays the packet statistics of all classes that are configured for all service policies either on the specified interface or subinterface or on a specific permanent virtual circuit (PVC) on the interface.
Step 6	`Router# exit`	(Optional) Exits privileged EXEC mode.

Modular QoS CLI Configuration Examples

This section provides the Modular QoS CLI configuration examples:

- Traffic Classes Defined Example
- Traffic Policy Created Example
- Traffic Policy Attached to an Interface Example
- match not Command Example

- Default Traffic Class Configuration Example
- class-map match-any and class-map match-all Commands Example
- Traffic Class as a Match Criterion (Nested Class Maps) Example
- Traffic Policy as a QoS Policy (Hierarchical Traffic Policies) Example

For information on how to configure the QoS functionality with the Modular QoS CLI, see the section "Modular QoS CLI Configuration Task List" in this chapter.

Traffic Classes Defined Example

In the following example, two traffic classes are created and their match criteria are defined. For the first traffic class called class1, access control list (ACL) 101 is used as the match criterion. For the second traffic class called class2, ACL 102 is used as the match criterion. Packets are checked against the contents of these ACLs to determine if they belong to the class.

```
Router(config)# class-map class1
Router(config-cmap)# match access-group 101
Router(config-cmap)# exit

Router(config)# class-map class2
Router(config-cmap)# match access-group 102
Router(config-cmap)# exit
```

Traffic Policy Created Example

In the following example, a traffic policy called policy1 is defined to contain policy specifications for the two classes—class1 and class2. The match criteria for these classes were defined in the traffic classes (see the section "Creating a Traffic Class" in this chapter).

For class1, the policy includes a bandwidth allocation request and a maximum packet count limit for the queue reserved for the class. For class2, the policy specifies only a bandwidth allocation request.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class1
Router(config-pmap-c)# bandwidth 3000
Router(config-pmap-c)# queue-limit 30
Router(config-pmap)# exit

Router(config-pmap)# class class2
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap)# exit
```

Traffic Policy Attached to an Interface Example

The following example shows how to attach an existing traffic policy (which was created in the preceding "Traffic Policy Created Example" section) to an interface. After you define a traffic policy with the **policy-map** command, you can attach it to one or more interfaces by using the **service-policy** command in interface configuration mode. Although you can assign the same traffic policy to multiple interfaces, each interface can have only one traffic policy attached at the input and only one traffic policy attached at the output.

```
Router(config)# interface e1/1
Router(config-if)# service-policy output policy1
Router(config-if)# exit

Router(config)# interface fa1/0/0
Router(config-if)# service-policy output policy1
Router(config-if)# exit
```

match not Command Example

The **match not** command is used to specify a specific QoS policy value that is not used as a match criterion. When using the **match not** command, all other values of that QoS policy become successful match criteria.

For instance, if the **match not qos-group 4** command is issued in class-map configuration mode, the specified class will accept all QoS group values except 4 as successful match criteria.

In the following traffic class, all protocols except IP are considered successful match criteria:

```
Router(config)# class-map noip
Router(config-cmap)# match not protocol ip
Router(config-cmap)# exit
```

Default Traffic Class Configuration Example

Unclassified traffic (traffic that does not meet the match criteria specified in the traffic classes) is treated as belonging to the default traffic class.

If the user does not configure a default class, packets are still treated as members of the default class. However, by default, the default class has no QoS features enabled. Therefore, packets belonging to a default class have no QoS functionality. These packets are placed into a first-in, first-out (FIFO) queue managed by tail drop. (Tail drop is a means of avoiding congestion that treats all traffic equally and does not differentiate between classes of service. Queues fill during periods of congestion. When the output queue is full and tail drop is in effect, packets are dropped until the congestion is eliminated and the queue is no longer full).

The following example configures a traffic policy for the default class of the traffic policy called policy1. The default class (which is always called class-default) has these characteristics: 10 queues for traffic that does not meet the match criteria of other classes whose policy is defined by the traffic policy policy1, and a maximum of 20 packets per queue before tail drop is enacted to handle additional queued packets.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class-default
Router(config-pmap-c)# fair-queue 10
Router(config-pmap-c)# queue-limit 20
```

For moredetailed information on the preceding commands, refer to the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T.

class-map match-any and class-map match-all Commands Example

This section illustrates the difference between the **class-map match-any** command and the **class-map match-all** command. The **match-any** and **match-all** options determine how packets are evaluated when multiple match criteria exist. Packets must either meet all of the match criteria (**match-all**) or one of the match criterion (**match-any**) in order to be considered a member of the traffic class.

The following example shows a traffic class configured with the **class-map match-all** command:

```
Router(config)# class-map match-all cisco1
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

If a packet arrives on a router with traffic class called cisco1 configured on the interface, the packet is evaluated to determine if it matches the IP protocol, QoS group 4, *and* access group 101. If all three of these match criteria are met, the packet matches traffic class cisco1.

The following example shows a traffic class configured with the **class-map match-any** command:

```
Router(config)# class-map match-any cisco2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

In traffic class called cisco2, the match criteria are evaluated consecutively until a successful match criterion is located. The packet is first evaluated to the determine whether IP protocol can be used as a match criterion. If IP protocol can be used as a match criterion, the packet is matched to traffic class cisco2. If IP protocol is not a successful match criterion, then QoS group 4 is evaluated as a match criterion. Each criterion is evaluated to see if the packet matches that criterion. Once a successful match occurs, the packet is classified as a member of traffic class cisco2. If the packet matches none of the specified criteria, the packet is classified as a member of the traffic class.

Note that the **class-map match-all** command requires that all of the match criteria must be met in order for the packet to be considered a member of the specified traffic class (a logical AND operator). In the example, protocol IP AND QoS group 4 AND access group 101 have to be successful match criteria. However, only one match criterion must be met for the packet in the **class-map match-any** command to be classified as a member of the traffic class (a logical OR operator). In the example, protocol IP OR QoS group 4 OR access group 101 have to be successful match criterion.

Traffic Class as a Match Criterion (Nested Class Maps) Example

There are two reasons to use the **match class-map** command. One reason is maintenance; if a large traffic class currently exists, using the traffic class match criterion is simply easier than retyping the same traffic class configuration.

The more common reason for the **match class-map** command is to allow users to use match-any and match-all statements in the same traffic class. If you want to combine match-all and match-any characteristics in a traffic policy, create a traffic class using one match criterion evaluation instruction (either match any or match all) and then use this traffic class as a match criterion in a traffic class that uses a different match criterion type.

Here is a possible scenario: Suppose A, B, C, and D were all separate match criterion, and you wanted traffic matching A, B, or C and D (A or B or [C and D]) to be classified as belonging to the traffic class. Without the nested traffic class, traffic would either have to match all 4 of the match criterion (A and B and C and D) or match any of the match criterion (A or B or C or D) to be considered part of the traffic class. You would not be able to combine "and" (match-all) and "or" (match-any) statements within the traffic class, and you would therefore be unable to configure the desired configuration.

The solution: Create one traffic class using match-all for C and D (which we will call criterion E), and then create a new match-any traffic class using A, B, and E. The new traffic class would have the correct evaluation sequence (A or B or E, which would also be A or B or [C and D]). The desired traffic class configuration has been achieved.

The only method of mixing match-all and match-any statements in a traffic class is through the use of the traffic class match criterion.

Nested Traffic Class for Maintenance Example

In the following example, the traffic class called class1 has the same characteristics as traffic class called class2, with the exception that traffic class class1 has added a destination address as a match criterion. Rather than configuring traffic class class1 line by line, you can enter the **match class-map class2** command. This command allows all of the characteristics in the traffic class called class2 to be included in the traffic class called class1, and you can simply add the new destination address match criterion without reconfiguring the entire traffic class.

```
Router(config)# class-map match-any class2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 3
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit

Router(config)# class-map match-all class1
Router(config-cmap)# match class-map class2
Router(config-cmap)# match destination-address mac 00.00.00.00.00.00
Router(config-cmap)# exit
```

Nested Traffic Class to Combine match-any and match-all Characteristics in One Traffic Class Example

The only method of including both match-any and match-all characteristics in a single traffic class is to use the **match class-map** command. To combine match-any and match-all characteristics into a single class, a traffic class created with the match-any instruction must use a class configured with the match-all instruction as a match criterion (through the **match class-map** command), or vice versa.

The following example shows how to combine the characteristics of two traffic classes, one with match-any and one with match-all characteristics, into one traffic class with the **match class-map** command. The result of traffic class class3 requires a packet to match one of the following three match criteria to be considered a member of traffic class class4: IP protocol *and* QoS group 4, destination MAC address 00.00.00.00.00.00, or access group 2.

In this example, only the traffic class called class4 is used with the traffic policy called policy1.

```
Router(config)# class-map match-all class3
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# exit

Router(config)# class-map match-any class4
Router(config-cmap)# match class-map class3
```

```
Router(config-cmap)# match destination-address mac 00.00.00.00.00.00
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit

Router(config)# policy-map policy1
Router(config-pmap)# class class4
Router(config-pmap-c)# police 8100 1500 2504 conform-action transmit exceed-action
set-qos-transmit 4
Router(config-pmap-c)# exit
```

Traffic Policy as a QoS Policy (Hierarchical Traffic Policies) Example

A traffic policy can be nested within a QoS policy when the **service-policy** command is used in policy-map class configuration mode. A traffic policy that contains a nested traffic policy is called a hierarchical traffic policy.

A hierarchical traffic policy contains a child and a parent policy. The child policy is the previously defined traffic policy that is being associated with the new traffic policy through the use of the **service-policy** command. The new traffic policy using the preexisting traffic policy is the parent policy. In the example in this section, traffic policy called child is the child policy and traffic policy called parent is the parent policy.

Hierarchical traffic policies can be attached to subinterfaces, Frame Relay PVCs, and ATM PVCs. A hierarchical traffic policy is particularly beneficial when configuring VIP-based distributed FRF.12 (and higher) PVCs. When hierarchical traffic policies are used, a single traffic policy (with a child and a parent policy) can be used to shape and prioritize PVC traffic. In the following example, the child policy is responsible for prioritizing traffic and the parent policy is responsible for shaping traffic. In this configuration, the parent policy allows packets to be sent from the interface, and the child policy determines the order in which the packets are sent.

```
Router(config)# policy-map child
Router(config-pmap)# class voice
Router(config-pmap-c)# priority 50

Router(config)# policy-map parent
Router(config-pmap)# class class-default
Router(config-pmap-c)# shape average 10000000
Router(config-pmap-c)# service-policy child
```

With the exception that the values associated with the **priority** and **shape** commands can be modified, the example is the required configuration for PVCs using FRF.12 (or higher). The value used with the **shape** command is provisioned from the committed information rate (CIR) value from the service provider. For more information about FRF.12 (or higher) PVCs, see the *Cisco IOS Wide-Area Networking Configuration Guide*, Release 12.4.

For more information about the **service-policy** command, see the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T.