Implementing Routing Policy

A routing policy instructs the router to inspect routes, filter them, and potentially modify their attributes as they are accepted from a peer, advertised to a peer, or redistributed from one routing protocol to another. This module describes how routing protocols make decisions to advertise, aggregate, discard, distribute, export, hold, import, redistribute and modify the routes based on configured routing policy.

The routing policy language (RPL) provides a single, straightforward language in which all routing policy needs can be expressed. RPL was designed to support large-scale routing configurations. It greatly reduces the redundancy inherent in previous routing policy configuration methods. RPL streamlines the routing policy configuration, reduces system resources required to store and process these configurations, and simplifies troubleshooting.

For more information about routing policy on the Cisco IOS XR software and complete descriptions of the routing policy commands listed in this module, see the Related Documents, on page 83 section of this module. To locate documentation for other commands that might appear while performing a configuration task, search online in the Cisco IOS XR Commands Master List for the Cisco CRS Router.

Feature History for Implementing Routing Policy

<table>
<thead>
<tr>
<th>Release</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release 2.0</td>
<td>This feature was introduced.</td>
</tr>
<tr>
<td>Release 3.0</td>
<td>No modification.</td>
</tr>
<tr>
<td>Release 3.2</td>
<td>No modification.</td>
</tr>
<tr>
<td>Release 3.3.0</td>
<td>Support was added for EIGRP, RIP policy, clear-policy, debug, OSPF area-in, and OSPF area-out attach points.</td>
</tr>
<tr>
<td>Release 3.4.0</td>
<td>Support was added for the BGP next-hop attach point. Support was also added for null sets and global parameterization.</td>
</tr>
<tr>
<td>Release</td>
<td>Modification</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Release 3.5.0</td>
<td>The following features were added:</td>
</tr>
<tr>
<td></td>
<td>• GNU Nano text editor for editing RPL policies.</td>
</tr>
<tr>
<td></td>
<td>• Enhanced prefix match functionality.</td>
</tr>
<tr>
<td></td>
<td>• Parameterization at attach points</td>
</tr>
<tr>
<td></td>
<td>• New 'done' disposition policy statement type</td>
</tr>
<tr>
<td>Release 3.6.0</td>
<td>No modification.</td>
</tr>
<tr>
<td>Release 3.7.0</td>
<td>Support was added for IS-IS inter-area-propagate attach point,</td>
</tr>
<tr>
<td></td>
<td>OSPF spf-prefix-priority attach point, and PIM policy rpf-topology attach</td>
</tr>
<tr>
<td></td>
<td>point.</td>
</tr>
<tr>
<td>Release 3.8.0</td>
<td>The vrf vrf-name keyword and argument were added for the</td>
</tr>
<tr>
<td></td>
<td>PIM policy rpf-topology attach point to support MVPN extranet.</td>
</tr>
<tr>
<td></td>
<td>The prefix matching performance feature was enhanced to use Radix trie</td>
</tr>
<tr>
<td></td>
<td>lookup.</td>
</tr>
<tr>
<td>Release 3.9.0</td>
<td>Parameterization was supported at all attach points.</td>
</tr>
<tr>
<td>Release 4.2.0</td>
<td>The following features were added:</td>
</tr>
<tr>
<td></td>
<td>• Hierarchical Conditions</td>
</tr>
<tr>
<td></td>
<td>• Apply Condition Policies</td>
</tr>
<tr>
<td>Release 4.2.1</td>
<td>The following features were introduced:</td>
</tr>
<tr>
<td></td>
<td>• Nested Wildcard Apply Policy.</td>
</tr>
<tr>
<td></td>
<td>• Editing Routing Policy Language set elements Using XML.</td>
</tr>
<tr>
<td></td>
<td>• Support 'set' as a valid operator for the 'med' attribute at the</td>
</tr>
<tr>
<td></td>
<td>bgp export and bgp import attach points.</td>
</tr>
</tbody>
</table>

- Prerequisites for Implementing Routing Policy, page 3
- Restrictions for Implementing Routing Policy, page 3
- Information About Implementing Routing Policy, page 3
- How to Implement Routing Policy, page 75
- Configuration Examples for Implementing Routing Policy, page 79
- Additional References, page 83
Prerequisites for Implementing Routing Policy

The following are prerequisites for implementing Routing Policy on Cisco IOS XR Software:

- You must be in a user group associated with a task group that includes the proper task IDs. The command reference guides include the task IDs required for each command. If you suspect user group assignment is preventing you from using a command, contact your AAA administrator for assistance.

- Border Gateway Protocol (BGP), integrated Intermediate System-to-Intermediate System (IS-IS), or Open Shortest Path First (OSPF) must be configured in your network.

Restrictions for Implementing Routing Policy

These restrictions apply when working with Routing Policy Language implementation on Cisco IOS XR software:

- An individual policy definition of up to 1000 statements are supported. The total number of statements within a policy can be extended to 4000 statements using hierarchical policy constructs. However, this limit is restricted with the use of `apply` statements.

- When a policy that is attached directly or indirectly to an attach point needs to be modified, a single `commit` operation cannot be performed when:
  - Removing a set or policy referred by another policy that is attached to any attach point directly or indirectly.
  - Modifying the policy to remove the reference to the same set or policy that is getting removed.

The `commit` must be performed in two steps:

1. Modify the policy to remove the reference to the policy or set and then `commit`.
2. Remove the policy or set and `commit`.

Information About Implementing Routing Policy

To implement RPL, you need to understand the following concepts:

Routing Policy Language

This section contains the following information:

Routing Policy Language Overview

RPL was developed to support large-scale routing configurations. RPL has several fundamental capabilities that differ from those present in configurations oriented to traditional route maps, access lists, and prefix lists. The first of these capabilities is the ability to build policies in a modular form. Common blocks of policy can
be defined and maintained independently. These common blocks of policy can then be applied from other blocks of policy to build complete policies. This capability reduces the amount of configuration information that needs to be maintained. In addition, these common blocks of policy can be parameterized. This parameterization allows for policies that share the same structure but differ in the specific values that are set or matched against to be maintained as independent blocks of policy. For example, three policies that are identical in every way except for the local preference value they set can be represented as one common parameterized policy that takes the varying local preference value as a parameter to the policy.

The policy language introduces the notion of sets. Sets are containers of similar data that can be used in route attribute matching and setting operations. Four set types exist: prefix-sets, community-sets, as-path-sets, and extcommunity-sets. These sets hold groupings of IPv4 or IPv6 prefixes, community values, AS path regular expressions, and extended community values, respectively. Sets are simply containers of data. Most sets also have an inline variant. An inline set allows for small enumerations of values to be used directly in a policy rather than having to refer to a named set. Prefix lists, community lists, and AS path lists must be maintained even when only one or two items are in the list. An inline set in RPL allows the user to place small sets of values directly in the policy body without having to refer to a named set.

Decision making, such as accept and deny, is explicitly controlled by the policy definitions themselves. RPL combines matching operators, which may use set data, with the traditional Boolean logic operators AND, OR, and NOT into complex conditional expressions. All matching operations return a true or false result. The execution of these conditional expressions and their associated actions can then be controlled by using simple if then, elseif, and else structures, which allow the evaluation paths through the policy to be fully specified by the user.

Routing Policy Language Structure

This section describes the basic structure of RPL.

Names

The policy language provides two kinds of persistent, namable objects: sets and policies. Definition of these objects is bracketed by beginning and ending command lines. For example, to define a policy named test, the configuration syntax would look similar to the following:

```
route-policy test
  [ . . . policy statements . . . ]
end-policy
```

Legal names for policy objects can be any sequence of the upper- and lowercase alphabetic characters; the numerals 0 to 9; and the punctuation characters period, hyphen, and underscore. A name must begin with a letter or numeral.

Sets

In this context, the term set is used in its mathematical sense to mean an unordered collection of unique elements. The policy language provides sets as a container for groups of values for matching purposes. Sets are used in conditional expressions. The elements of the set are separated by commas. Null (empty) sets are allowed.

In the following example:

```
prefix-set backup-routes
  # currently no backup routes are defined
```
end-set

a condition such as:

    if destination in backup-routes then

evaluates as FALSE for every route, because there is no match-condition in the prefix set that it satisfies.

Five kinds of sets exist: as-path-set, on page 5, community-set, on page 6, extcommunity-set, on page 7, prefix-set, on page 10, and rd-set, on page 11. You may want to perform comparisons against a small number of elements, such as two or three community values, for example. To allow for these comparisons, the user can enumerate these values directly. These enumerations are referred to as inline sets. Functionally, inline sets are equivalent to named sets, but allow for simple tests to be inline. Thus, comparisons do not require that a separate named set be maintained when only one or two elements are being compared. See the set types described in the following sections for the syntax. In general, the syntax for an inline set is a comma-separated list surrounded by parentheses as follows: (element-entry , element-entry , element-entry, ...element-entry), where element-entry is an entry of an item appropriate to the type of usage such as a prefix or a community value.

The following is an example using an inline community set:

    route-policy sample-inline
    if community matches-any ([10..15]:100) then
    set local-preference 100
    endif
    end-policy

The following is an equivalent example using the named set test-communities:

    community-set test-communities
    10:100,
    11:100,
    12:100,
    13:100,
    14:100,
    15:100
    end-set

    route-policy sample
    if community matches-any test-communities then
    set local-preference 100
    endif
    end-policy

Both of these policies are functionally equivalent, but the inline form does not require the configuration of the community set just to store the six values. You can choose the form appropriate to the configuration context. In the following sections, examples of both the named set version and the inline form are provided where appropriate.

**as-path-set**

An AS path set comprises operations for matching an AS path attribute. The matching operations are:

- dfa-regex: DFA style regular expression
- ios-regex: Traditional IOS style regular expression
- length: Length of BGP AS-path
• neighbor-is: BGP AS-path neighbor is
• originates-from: BGP AS-path originates-from
• passes-through: BGP AS-path passes-through
• unique-length: Length of BGP AS-path ignoring duplicates

Named Set Form

The named set form uses the **ios-regex** keyword to indicate the type of regular expression and requires single quotation marks around the regular expression.

The following is a sample definition of a named AS path set:

```
as-path-set aset1
  ios-regex '_42$'
  ios-regex '_127$'
end-set
```

This AS path set comprises two elements. When used in a matching operation, this AS path set matches any route whose AS path ends with either the autonomous system (AS) number 42 or 127.

To remove the named AS path set, use the **no as-path-set aset1** command-line interface (CLI) command.

---

**Note**

Regular expression matching is CPU intensive. The policy performance can be substantially improved by either collapsing the regular expression patterns together to reduce the total number of regular expression invocations or by using equivalent native as-path match operations such as ‘as-path neighbor-is’, ‘as-path originates-from’ or ‘as-path passes-through’.

---

Inline Set Form

The inline set form is a parenthesized list of comma-separated expressions, as follows:

```
(ios-regex '_42$', ios-regex '_127$')
```

This set matches the same AS paths as the previously named set, but does not require the extra effort of creating a named set separate from the policy that uses it.

community-set

A community-set holds community values for matching against the BGP community attribute. A community is a 32-bit quantity. Integer community values **must** be split in half and expressed as two unsigned decimal integers in the range from 0 to 65535, separated by a colon. Single 32-bit community values are not allowed. The following is the named set form:

Named Set Form

```
community-set cset1
  12:34,
  12:56,
  12:78,
  internet
```
The inline form of a community-set also supports parameterization. Each 16-bit portion of the community may be parameterized. See the Parameterization, on page 15 for more information.

RPL provides symbolic names for the standard well-known community values: internet is 0:0, no-export is 65535:65281, no-advertise is 65535:65282, and local-as is 65535is-empty:65283.

RPL also provides a facility for using wildcards in community specifications. A wildcard is specified by inserting an asterisk (*) in place of one of the 16-bit portions of the community specification; the wildcard indicates that any value for that portion of the community matches. Thus, the following policy matches all communities in which the autonomous system part of the community is 123:

```
 community-set cset3
  123:*
 end-set
```

A community set can either be empty, or contain one or more community values. When used with an empty community set, the is-empty operator will evaluate to TRUE and the matches-any and matches-every operators will evaluate to FALSE.

### extcommunity-set

An extended community-set is analogous to a community-set except that it contains extended community values instead of regular community values. It also supports named forms and inline forms. There are three types of extended community sets: cost, soo, and rt.

As with community sets, the inline form supports parameterization within parameterized policies. Either portion of the extended community value can be parameterized.

Wildcards (*) and regular expressions are allowed for extended community set elements.

Every extended community-set must contain at least one extended community value. Empty extended community-sets are invalid and rejected.

The following are syntactic examples:

#### Named Form for Extcommunity-set Cost

A cost set is an extcommunity set used to store cost EIGRP Cost Community type extended community type communities.

```
 extcommunity-set cost a_cost_set
  IGP:1:10
 end-set
```

These options are supported under extended community set Cost:

```
RP/0/RP0/CPU0:router(config)# extcommunity-set cost cost_set
RP/0/RP0/CPU0:router(config-ext)#?
    #-remark    Remark beginning with '
    <0-255>     decimal number
```
abort: Discard RPL definition and return to top level config
end-set: End of set definition
exit: Exit from this submode
igp: Cost Community with IGP as point of insertion
pre-bestpath: Cost Community with Pre-Bestpath as point of insertion
show: Show partial RPL configuration

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-remark</td>
<td>Remark beginning with '#'</td>
</tr>
<tr>
<td>&lt;0-255&gt;</td>
<td>decimal number</td>
</tr>
<tr>
<td>abort</td>
<td>Discard RPL definition and return to top level config</td>
</tr>
<tr>
<td>end-set</td>
<td>End of set definition</td>
</tr>
<tr>
<td>exit</td>
<td>Exit from this submode</td>
</tr>
<tr>
<td>igp:</td>
<td>Cost Community with IGP as point of insertion</td>
</tr>
<tr>
<td>pre-bestpath:</td>
<td>Cost Community with Pre-Bestpath as point of insertion</td>
</tr>
<tr>
<td>show</td>
<td>Show partial RPL configuration</td>
</tr>
</tbody>
</table>

**Named Form for Extcommunity-set RT**

An rt set is an extcommunity set used to store BGP Route Target (RT) extended community type communities:

```
extcommunity-set rt a_rt_set
1.2.3.4:666
1234:666,
1.2.3.4:777,
4567:777
end-set
```

**Inline Set Form for Extcommunity-set RT**

```
(1.2.3.4:666, 1234:666, 1.2.3.4:777, 4567:777)
($ipadrr:666, 1234:$tag, 1.2.3.4:777, $tag2:777)
```

These options are supported under extended community set RT:

```
RP/0/RP0/CPU0:router(config)#extcommunity-set rt rt_set
RP/0/RP0/CPU0:router(config-ext)#
#-remark: Remark beginning with '#'
*: Wildcard (any community or part thereof)
<1-4294967295>: 32-bit decimal number
<1-65535>: 16-bit decimal number
A.B.C.D/M:N: Extended community - IPv4 prefix format
A.B.C.D:N: Extended community - IPv4 format
ASN:N: Extended community - ASPLAIN format
X.Y:N: Extended community - ASDOT format
abort: Discard RPL definition and return to top level config
dfa-regex: DFA style regular expression
dfa-regex: DFA style regular expression
dfa-regex: DFA style regular expression
end-set: End of set definition
exit: Exit from this submode
ios-regex: Traditional IOS style regular expression
show: Show partial RPL configuration
```
Named Form for Extcommunity-set Soo

A soo set is an extcommunity set used to store BGP Site-of-Origin (SoO) extended community type communities:

```
extcommunity-set soo a_soo_set
1.1.1:100,
   100:200
end-set
```

These options are supported under extended community set Soo:

```
RP/0/RP0/CPU0:router(config-ext)#extcommunity-set soo soo_set
#-remark Remark beginning with '#'
* Wildcard (any community or part thereof)
<1-4294967295> 32-bit decimal number
<1-65535> 16-bit decimal number
A.B.C.D/M:N Extended community - IPv4 prefix format
A.B.C.D:N Extended community - IPv4 format
ASN:N Extended community - ASPLAIN format
X.Y:N Extended community - ASDOT format
abort Discard RPL definition and return to top level config
dfa-regex DFA style regular expression
end-set End of set definition
exit Exit from this submode
ios-regex Traditional IOS style regular expression
show Show partial RPL configuration
```

Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x
A prefix-set holds IPv4 or IPv6 prefix match specifications, each of which has four parts: an address, a mask length, a minimum matching length, and a maximum matching length. The address is required, but the other three parts are optional. The address is a standard dotted-decimal IPv4 or colon-separated hexadecimal IPv6 address. The mask length, if present, is a nonnegative decimal integer in the range from 0 to 32 (0 to 128 for IPv6) following the address and separated from it by a slash. The optional minimum matching length follows the address and optional mask length and is expressed as the keyword `ge` (mnemonic for greater than or equal to), followed by a nonnegative decimal integer in the range from 0 to 32 (0 to 128 for IPv6). The optional maximum matching length follows the rest and is expressed by the keyword `le` (mnemonic for less than or equal to), followed by yet another nonnegative decimal integer in the range from 0 to 32 (0 to 128 for IPv6). A syntactic shortcut for specifying an exact length for prefixes to match is the `eq` keyword (mnemonic for equal to).

If a prefix match specification has no mask length, then the default mask length is 32 for IPv4 and 128 for IPv6. The default minimum matching length is the mask length. If a minimum matching length is specified, then the default maximum matching length is 32 for IPv4 and 128 for IPv6. Otherwise, if neither minimum nor maximum is specified, the default maximum is the mask length.

Radix trie lookup is used to perform prefix-set matching.

The prefix-set itself is a comma-separated list of prefix match specifications. The following are examples:

```
prefix-set legal-ipv4-prefix-examples
  10.0.1.1,
  10.0.2.0/24,
  10.0.3.0/24 ge 28,
  10.0.4.0/24 le 28,
  10.0.5.0/24 ge 26 le 30,
  10.0.6.0/24 eq 28,
  10.0.7.2/32 ge 16 le 24,
  10.0.8.0/26 ge 8 le 16
end-set
```
The first element of the prefix-set matches only one possible value, 10.0.1.1/32 or the host address 10.0.1.1. The second element matches only one possible value, 10.0.2.0/24. The third element matches a range of prefix values, from 10.0.3.0/24 to 10.0.3.255/32. The fourth element matches a range of values, from 10.0.4.0/24 to 10.0.4.240/28. The fifth element matches prefixes in the range from 10.0.5.0/26 to 10.0.5.252/30. The sixth element matches any prefix of length 28 in the range from 10.0.6.0/28 through 10.0.6.240/28. The seventh element matches any prefix of length 32 in the range 10.0.0.0.2/32 to 10.0.255.2/32. The eighth element matches any prefix of length 32 in the range 10.0.0.0.2/32 to 10.0.255.2/32.

The following prefix-set consists entirely of invalid prefix match specifications:

```
prefix-set ILLEGAL-PREFIX-EXAMPLES
  10.1.1.1 ge 16,
  10.1.2.1 le 16,
  10.1.3.0/24 ge 23,
  10.1.4.0/24 ge 33,
  10.1.5.0/25 ge 29 le 28
end-set
```

Neither the minimum length nor maximum length is valid without a mask length. For IPv4, the minimum length must be less than 32, the maximum length of an IPv4 prefix. For IPv6, the minimum length must be less than 128, the maximum length of an IPv6 prefix. The maximum length must be equal to or greater than the minimum length.

**rd-set**

An rd-set is used to create a set with route distinguisher (RD) elements. An RD set is a 64-bit value prepended to an IPv4 address to create a globally unique Border Gateway Protocol (BGP) VPN IPv4 address.

You can define RD values with the following commands:

- `a.b.c.d:m:*`—BGP VPN RD in IPv4 format with a wildcard character. For example, 10.0.0.2:255.255.0.0:*
- `a.b.c.d/m:n`—BGP VPN RD in IPv4 format with a mask. For example, 10.0.0.2:255.255.0.0:666.
- `a.b.c.d:**`—BGP VPN RD in IPv4 format with a wildcard character. For example, 10.0.0.2:255.255.0.0.
- `a.b.c.d:n`—BGP VPN RD in IPv4 format. For example, 10.0.0.2:666.
- `asn:*`—BGP VPN RD in ASN format with a wildcard character. For example, 10002:255.255.0.0.
- `asn:n`—BGP VPN RD in ASN format. For example, 10002:666.

The following is an example of an rd-set:

```
rd-set rdset1
  10.0.0.0/8:*,
  10.0.0.0/8:777,
  10.0.0.0:*,
  10.0.0.0:777,
  65000:*,
```
Routing Policy Language Components

Four main components in the routing policy language are involved in defining, modifying, and using policies: the configuration front end, policy repository, execution engine, and policy clients themselves.

The configuration front end (CLI) is the mechanism to define and modify policies. This configuration is then stored on the router using the normal storage means and can be displayed using the normal configuration show commands.

The second component of the policy infrastructure, the policy repository, has several responsibilities. First, it compiles the user-entered configuration into a form that the execution engine can understand. Second, it performs much of the verification of policies; and it ensures that defined policies can actually be executed properly. Third, it tracks which attach points are using which policies so that when policies are modified the appropriate clients are properly updated with the new policies relevant to them.

The third component is the execution engine. This component is the piece that actually runs policies as the clients request. The process can be thought of as receiving a route from one of the policy clients and then executing the actual policy against the specific route data.

The fourth component is the policy clients (the routing protocols). This component calls the execution engine at the appropriate times to have a given policy be applied to a given route, and then perform some number of actions. These actions may include deleting the route if policy indicated that it should be dropped, passing along the route to the protocol decision tree as a candidate for the best route, or advertising a policy modified route to a neighbor or peer as appropriate.

Routing Policy Language Usage

This section provides basic routing policy language usage examples. See the How to Implement Routing Policy, on page 75 for detailed information on how to implement routing policy language.

Pass Policy

The following example shows how the policy accepts all presented routes without modifying the routes.

```
route-policy quickstart-pass
pass
end-policy
```

Drop Everything Policy

The following example shows how the policy explicitly rejects all routes presented to it. This type of policy is used to ignore everything coming from a specific peer.

```
route-policy quickstart-drop
drop
end-policy
```
Ignore Routes with Specific AS Numbers in the Path

The following example shows the policy definition in three parts. First, the `as-path-set` command defines three regular expressions to match against an AS path. Second, the `route-policy` command applies the AS path set to a route. If the AS path attribute of the route matches the regular expression defined with the `as-path-set` command, the protocol refuses the route. Third, the route policy is attached to BGP neighbor 10.0.1.2. BGP consults the policy named `ignore_path` on routes received (imported) from neighbor 10.0.1.2.

```conf
as-path-set ignore_path
  ios-regex '_11_',
  ios-regex '_22_',
  ios-regex '_33_'
end-set

route-policy ignore_path_as
  if as-path in ignore_path then
    drop
  else
    pass
  endif
end-policy

router bgp 2
  neighbor 10.0.1.2 address-family ipv4 unicast policy ignore_path_as in
```

Set Community Based on MED

The following example shows how the policy tests the MED of a route and modifies the community attribute of the route based on the value of the MED. If the MED value is 127, the policy adds the community 123:456 to the route. If the MED value is 63, the policy adds the value 123:789 to the community attribute of the route. Otherwise, the policy removes the community 123:123 from the route. In any case, the policy instructs the protocol to accept the route.

```conf
route-policy quickstart-med
  if med eq 127 then
    set community (123:456) additive
  elseif med eq 63 then
    set community (123:789) additive
  else
    delete community in (123:123)
  endif
  pass
end-policy
```

Set Local Preference Based on Community

The following example shows how the community-set named `quickstart-communities` defines community values. The route policy named `quickstart-localpref` tests a route for the presence of the communities specified in the `quickstart-communities` community set. If any of the community values are present in the route, the route policy sets the local preference attribute of the route to 31. In any case, the policy instructs the protocol to accept the route.

```conf
community-set quickstart-communities
  987:654,
  987:543,
  987:321,
  987:210
end-set

route-policy quickstart-localpref
  if community matches-any quickstart-communities then
```
Persistent Remarks

The following example shows how comments are placed in the policy to clarify the meaning of the entries in
the set and the statements in the policy. The remarks are persistent, meaning they remain attached to the policy.
For example, remarks are displayed in the output of the show running-config command. Adding remarks to
the policy makes the policy easier to understand, modify at a later date, and troubleshoot if an unexpected
behavior occurs.

```
prefix-set rfc1918
  # These are the networks defined as private in RFC1918 (including
  # all subnets thereof)
  10.0.0.0/8 ge 8,
  172.16.0.0/12 ge 12,
  192.168.0.0/16 ge 16
end-set
route-policy quickstart-remarks
  # Handle routes to RFC1918 networks
  if destination in rfc1918 then
    # Set the community such that we do not export the route
    set community (no-export) additive
  endif
end-policy
```

Routing Policy Configuration Basics

Route policies comprise series of statements and expressions that are bracketed with the route-policy and
end-policy keywords. Rather than a collection of individual commands (one for each line), the statements
within a route policy have context relative to each other. Thus, instead of each line being an individual
command, each policy or set is an independent configuration object that can be used, entered, and manipulated
as a unit.

Each line of a policy configuration is a logical subunit. At least one new line must follow the then, else,
and end-policy keywords. A new line must also follow the closing parenthesis of a parameter list and the
name string in a reference to an AS path set, community set, extended community set, or prefix set. At least
one new line must precede the definition of a route policy, AS path set, community set, extended community
set, or prefix set. One or more new lines can follow an action statement. One or more new lines can follow a
comma separator in a named AS path set, community set, extended community set, or prefix set. A new line
must appear at the end of a logical unit of policy expression and may not appear anywhere else.

Policy Definitions

Policy definitions create named sequences of policy statements. A policy definition consists of the CLI
route-policy keyword followed by a name, a sequence of policy statements, and the end-policy keyword.
For example, the following policy drops any route it encounters:

```
route-policy drop-everything
  drop
end-policy
```
The name serves as a handle for binding the policy to protocols. To remove a policy definition, issue the **no route-policy name** command.

Policies may also refer to other policies such that common blocks of policy can be reused. This reference to other policies is accomplished by using the **apply** statement, as shown in the following example:

```
route-policy check-as-1234
  if as-path passes-through '1234.5' then
    apply drop-everything
  else
    pass
  endif
end-policy
```

The **apply** statement indicates that the policy drop-everything should be executed if the route under consideration passed through autonomous system 1234.5 before it is received. If a route that has autonomous system 1234.5 in its AS path is received, the route is dropped; otherwise, the route is accepted without modification. This policy is an example of a hierarchical policy. Thus, the semantics of the **apply** statement are just as if the applied policy were cut and pasted into the applying policy:

```
route-policy check-as-1234-prime
  if as-path passes-through '1234.5' then
    drop
  else
    pass
  endif
end-policy
```

You may have as many levels of hierarchy as desired. However, many levels may be difficult to maintain and understand.

### Parameterization

In addition to supporting reuse of policies using the **apply** statement, policies can be defined that allow for parameterization of some of the attributes. The following example shows how to define a parameterized policy named param-example. In this case, the policy takes one parameter, $mytag. Parameters always begin with a dollar sign and consist otherwise of any alphanumeric characters. Parameters can be substituted into any attribute that takes a parameter.

In the following example, a 16-bit community tag is used as a parameter:

```
route-policy param-example ($mytag)
  set community (1234:$mytag) additive
end-policy
```

This parameterized policy can then be reused with different parameterization, as shown in the following example. In this manner, policies that share a common structure but use different values in some of their individual statements can be modularized. For details on which attributes can be parameterized, see the individual attribute sections.

```
route-policy origin-10
  if as-path originates-from '10.5' then
    apply param-example(10.5)
  else
```

---

*Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x*
The parameterized policy param-example provides a policy definition that is expanded with the values provided as the parameters in the apply statement. Note that the policy hierarchy is always maintained. Thus, if the definition of param-example changes, then the behavior of origin_10 and origin_20 changes to match.

The effect of the origin-10 policy is that it adds the community 1234:10 to all routes that pass through this policy and have an AS path indicating the route originated from autonomous system 10. The origin-20 policy is similar except that it adds to community 1234:20 for routes originating from autonomous system 20.

**Parameterization at Attach Points**

In addition to supporting parameterization using the apply statement described in the Parameterization, on page 15, policies can also be defined that allow for parameterization the attributes at attach points. Parameterization is supported at all attach points.

In the following example, we define a parameterized policy "param-example". In this example, the policy takes two parameters "$mymed" and "$prefixset". Parameters always begin with a dollar sign, and consist otherwise of any alphanumeric characters. Parameters can be substituted into any attribute that takes a parameter.

In this example we are passing a MED value and prefix set name as parameters.

```plaintext
route-policy param-example ($mymed, $prefixset)
  if destination in $prefixset then
    set med $mymed
  endif
end-policy
```

This parameterized policy can then be reused with different parameterizations as shown in the example below. In this manner, policies that share a common structure but use different values in some of their individual statements can be modularized. For details on which attributes can be parameterized, see the individual attributes for each protocol.

```plaintext
router bgp 2
  neighbor 10.1.1.1
    remote-as 3
    address-family ipv4 unicast
      route-policy param-example(10, prefix_set1)
      route-policy param-example(20, prefix_set2)
```

The parameterized policy param-example provides a policy definition that is expanded with the values provided as the parameters in the neighbor route-policy in and out statement.
Global Parameterization

RPL supports the definition of systemwide global parameters that can be used inside policy definition. Global parameters can be configured as follows:

```
Policy-global
  glbpathtype 'ebgp'
  glbtag '100'
end-global
```

The global parameter values can be used directly inside a policy definition similar to the local parameters of parameterized policy. In the following example, the `globalparam` argument, which makes use of the global parameters `gbpathtype` and `glbtag`, is defined for a nonparameterized policy.

```
route-policy globalparam
  if path-type is $glbpathtype then
    set tag $glbtag
  endif
end-policy
```

When a parameterized policy has a parameter name "collision" with a global parameter name, parameters local to policy definition take precedence, effectively masking off global parameters. In addition, a validation mechanism is in place to prevent the deletion of a particular global parameter if it is referred by any policy.

Semantics of Policy Application

This section discusses how routing policies are evaluated and applied. The following concepts are discussed:

Boolean Operator Precedence

Boolean expressions are evaluated in order of operator precedence, from left to right. The highest precedence operator is NOT, followed by AND, and then OR. The following expression:

```
med eq 10 and not destination in (10.1.3.0/24) or community matches-any ([10..25]:35)
```

if fully parenthesized to display the order of evaluation, would look like this:

```
(med eq 10 and (not destination in (10.1.3.0/24))) or community matches-any ([10..25]:35)
```

The inner NOT applies only to the destination test; the AND combines the result of the NOT expression with the Multi Exit Discriminator (MED) test; and the OR combines that result with the community test. If the order of operations are rearranged:

```
not med eq 10 and destination in (10.1.3.0/24) or community matches-any ([10..25]:35)
```

then the expression, fully parenthesized, would look like the following:

```
((not med eq 10) and destination in (10.1.3.0/24)) or community matches-any ([10..25]:35)
```
Multiple Modifications of the Same Attribute

When a policy replaces the value of an attribute multiple times, the last assignment wins because all actions are executed. Because the MED attribute in BGP is one unique value, the last value to which it gets set to wins. Therefore, the following policy results in a route with a MED value of 12:

```plaintext
set med 9
set med 10
set med 11
set med 12
```

This example is trivial, but the feature is not. It is possible to write a policy that effectively changes the value for an attribute. For example:

```plaintext
set med 8
if community matches-any cs1 then
  set local-preference 122
if community matches-any cs2 then
  set med 12
endif
endif
```

The result is a route with a MED of 8, unless the community list of the route matches both cs1 and cs2, in which case the result is a route with a MED of 12.

In the case in which the attribute being modified can contain only one value, it is easy to think of this case as the last statement wins. However, a few attributes can contain multiple values and the result of multiple actions on the attribute is cumulative rather than as a replacement. The first of these cases is the use of the `additive` keyword on community and extended community evaluation. Consider a policy of the form:

```plaintext
route-policy community-add
set community (10:23)
set community (10:24) additive
set community (10:25) additive
end-policy
```

This policy sets the community string on the route to contain all three community values: 10:23, 10:24, and 10:25.

The second of these cases is AS path prepending. Consider a policy of the form:

```plaintext
route-policy prepend-example
prepend as-path 2.5 3
prepend as-path 666.5 2
end-policy
```

This policy prepends 666.5 666.5 2.5 2.5 2.5 to the AS path. This prepending is a result of all actions being taken and to the AS path being an attribute that contains an array of values rather than a simple scalar value.
When Attributes Are Modified

A policy does not modify route attribute values until all tests have been completed. In other words, comparison operators always run on the initial data in the route. Intermediate modifications of the route attributes do not have a cascading effect on the evaluation of the policy. Take the following example:

```plaintext
if med eq 12 then
    set med 42
if med eq 42 then
    drop
endif
endif
```

This policy never executes the drop statement because the second test (med eq 42) sees the original, unmodified value of the MED in the route. Because the MED has to be 12 to get to the second test, the second test always returns false.

Default Drop Disposition

All route policies have a default action to drop the route under evaluation unless the route has been modified by a policy action or explicitly passed. Applied (nested) policies implement this disposition as though the applied policy were pasted into the point where it is applied.

Consider a policy to allow all routes in the 10 network and set their local preference to 200 while dropping all other routes. You might write the policy as follows:

```plaintext
route-policy two
    if destination in (10.0.0.0/8 ge 8 le 32) then
        set local-preference 200
    endif
end-policy

route-policy one
    apply two
end-policy
```

It may appear that policy one drops all routes because it neither contains an explicit `pass` statement nor modifies a route attribute. However, the applied policy does set an attribute for some routes and this disposition is passed along to policy one. The result is that policy one passes routes with destinations in network 10, and drops all others.

Control Flow

Policy statements are processed sequentially in the order in which they appear in the configuration. Policies that hierarchically reference other policy blocks are processed as if the referenced policy blocks had been directly substituted inline. For example, if the following policies are defined:

```plaintext
route-policy one
    set weight 100
end-policy

route-policy two
    set med 200
end-policy

route-policy three
```
apply two
set community (2:666) additive
end-policy

route-policy four
apply one
apply three
pass
end-policy

Policy four could be rewritten in an equivalent way as follows:

route-policy four-equivalent
set weight 100
set med 200
set community (2:666) additive
pass
end-policy

---

**Note**
The *pass* statement is not required and can be removed to represent the equivalent policy in another way.

---

### Policy Verification

Several different types of verification occur when policies are being defined and used.

#### Range Checking

As policies are being defined, some simple verifications, such as range checking of values, is done. For example, the MED that is being set is checked to verify that it is in a proper range for the MED attribute. However, this range checking cannot cover parameter specifications because they may not have defined values yet. These parameter specifications are verified when a policy is attached to an attach point. The policy repository also verifies that there are no recursive definitions of policy, and that parameter numbers are correct. At attach time, all policies must be well formed. All sets and policies that they reference must be defined and have valid values. Likewise, any parameter values must also be in the proper ranges.

#### Incomplete Policy and Set References

As long as a given policy is not attached at an attach point, the policy is allowed to refer to nonexistent sets and policies, which allows for freedom of workflow. You can build configurations that reference sets or policy blocks that are not yet defined, and then can later fill in those undefined policies and sets, thereby achieving much greater flexibility in policy definition. Every piece of policy you want to reference while defining a policy need not exist in the configuration. Thus, a user can define a policy sample that references the policy bar using an *apply* statement even if the policy bar does not exist. Similarly, a user can enter a policy statement that refers to a nonexistent set.

However, the existence of all referenced policies and sets is enforced when a policy is attached. If you attempt to attach the policy sample with the reference to an undefined policy bar at an inbound BGP policy using the `neighbor 1.2.3.4 address-family ipv4 unicast policy sample in` command, the configuration attempt is rejected because the policy bar does not exist.
Likewise, you cannot remove a route policy or set that is currently in use at an attach point because this removal would result in an undefined reference. An attempt to remove a route policy or set that is currently in use results in an error message to the user.

A condition exists that is referred to as a null policy in which the policy bar exists but has no statements, actions, or dispositions in it. In other words, the policy bar does exist as follows:

```
route-policy bar
end-policy
```

This is a valid policy block. It effectively forces all routes to be dropped because it is a policy block that never modifies a route, nor does it include the pass statement. Thus, the default action of drop for the policy block is followed.

**Attached Policy Modification**

Policies that are in use do, on occasion, need to be modified. Traditionally, configuration changes are done by completely removing the relevant configuration and then re-entering it. However, this allows for a window of time in which no policy is attached and the default action takes place. RPL provides a mechanism for an atomic change so that if a policy is redeclared, or edited using a text editor, the new configuration is applied immediately—which allows for policies that are in use to be changed without having a window of time in which no policy is applied at the given attach point.

**Verification of Attribute Comparisons and Actions**

The policy repository knows which attributes, actions, and comparisons are valid at each attach point. When a policy is attached, these actions and comparisons are verified against the capabilities of that particular attach point. Take, for example, the following policy definition:

```
route-policy bad
set med 100
set level level-1-2
set ospf-metric 200
end-policy
```

This policy attempts to perform actions to set the BGP attribute med, IS-IS attribute level, and OSPF attribute cost. The system allows you to define such a policy, but it does not allow you to attach such a policy. If you had defined the policy bad and then attempted to attach it as an inbound BGP policy using the BGP configuration statement `neighbor 1.2.3.4 address-family ipv4 unicast route-policy bad in` the system would reject this configuration attempt. This rejection results from the verification process checking the policy and realizing that while BGP could set the MED, it has no way of setting the level or cost as the level and cost are attributes of IS-IS and OSPF, respectively. Instead of silently omitting the actions that cannot be done, the system generates an error to the user. Likewise, a valid policy in use at an attach point cannot be modified in such a way as to introduce an attempt to modify a nonexistent attribute or to compare against a nonexistent attribute. The verifiers test for nonexistent attributes and reject such a configuration attempt.

**Policy Statements**

Four types of policy statements exist: remark, disposition (drop and pass), action (set), and if (comparator).
Remark

A remark is text attached to policy configuration but otherwise ignored by the policy language parser. Remarks are useful for documenting parts of a policy. The syntax for a remark is text that has each line prepended with a pound sign (#):

```
# This is a simple one-line remark.
# This
# is a remark
# comprising multiple
# lines.
```

In general, remarks are used between complete statements or elements of a set. Remarks are not supported in the middle of statements or within an inline set definition.

Unlike traditional !-comments in the CLI, RPL remarks persist through reboots and when configurations are saved to disk or a TFTP server and then loaded back onto the router.

Disposition

If a policy modifies a route, by default the policy accepts the route. RPL provides a statement to force the opposite—the drop statement. If a policy matches a route and executes a drop, the policy does not accept the route. If a policy does not modify the route, by default the route is dropped. To prevent the route from being dropped, the pass statement is used.

The drop statement indicates that the action to take is to discard the route. When a route is dropped, no further execution of policy occurs. For example, if after executing the first two statements of a policy the drop statement is encountered, the policy stops and the route is discarded.

All policies have a default drop action at the end of execution.

The pass statement allows a policy to continue executing even though the route has not been modified. When a policy has finished executing, any route that has been modified in the policy or any route that has received a pass disposition in the policy, successfully passes the policy and completes the execution. If route policy B_r is applied within route policy A_r, execution continues from policy A_r to policy B_r and back to policy A_r provided prefix is not dropped by policy B_r.

```
route-policy A_r
    set community (10:10)
    apply B_r
end-policy
!

route-policy B_r
    if destination in (121.23.0.0/16 le 32, 155.12.0.0/16 le 32) then
        set community (121:155) additive
    endif
end-policy
!
```

By default, a route is dropped at the end of policy processing unless either the policy modifies a route attribute or it passes the route by means of an explicit pass statement. For example, if route-policy B is applied within
route-policy A, then execution continues from policy A to policy B and back to policy A, provided the prefix is not dropped by policy B.

```
routing-policy A
  if as-path neighbor-is '123' then
    apply B
  policy statement N
end-policy
```

Whereas the following policies pass all routes that they evaluate.

```
routing-policy PASS-ALL
  pass
end-policy
```

```
routing-policy SET-LPREF
  set local-preference 200
end-policy
```

In addition to being implicitly dropped, a route may be dropped by an explicit drop statement. Drop statements cause a route to be dropped immediately so that no further policy processing is done. Note also that a drop statement overrides any previously processed pass statements or attribute modifications. For example, the following policy drops all routes. The first pass statement is executed, but is then immediately overridden by the drop statement. The second pass statement never gets executed.

```
routing-policy DROP-EXAMPLE
  pass
  drop
  pass
end-policy
```

When one policy applies another, it is as if the applied policy were copied into the right place in the applying policy, and then the same drop-and-pass semantics are put into effect. For example, policies ONE and TWO are equivalent to policy ONE-PRIME:

```
routing-policy ONE
  apply two
  if as-path neighbor-is '123' then
    pass
  endif
end-policy
```

```
routing-policy TWO
  if destination in (10.0.0.0/16 le 32) then
    drop
  endif
end-policy
```

```
routing-policy ONE-PRIME
  if destination in (10.0.0.0/16 le 32) then
    drop
  endif
  if as-path neighbor-is '123' then
    pass
  endif
end-policy
```

Because the effect of an explicit drop statement is immediate, routes in 10.0.0.0/16 le 32 are dropped without any further policy processing. Other routes are then considered to see if they were advertised by autonomous
system 123. If they were advertised, they are passed; otherwise, they are implicitly dropped at the end of all policy processing.

The **done** statement indicates that the action to take is to stop executing the policy and accept the route. When encountering a **done** statement, the route is passed and no further policy statements are executed. All modifications made to the route prior to the **done** statement are still valid.

**Action**

An action is a sequence of primitive operations that modify a route. Most actions, but not all, are distinguished by the **set** keyword. In a route policy, actions can be grouped together. For example, the following is a route policy comprising three actions:

```
route-policy actions
set med 217
set community (12:34) additive
delete community in (12:56)
end-policy
```

**If**

In its simplest form, an **if** statement uses a conditional expression to decide which actions or dispositions should be taken for the given route. For example:

```
if as-path in as-path-set-1 then
drop
endif
```

The example indicates that any routes whose AS path is in the set as-path-set-1 are dropped. The contents of the **then** clause may be an arbitrary sequence of policy statements.

The following example contains two action statements:

```
if origin is igp then
set med 42
prepend as-path 73.5 5
endif
```

The CLI provides support for the **exit** command as an alternative to the **endif** command.

The **if** statement also permits an **else** clause, which is executed if the if condition is false:

```
if med eq 8 then
set community (12:34) additive
else
set community (12:56) additive
endif
```

The policy language also provides syntax, using the **elseif** keyword, to string together a sequence of tests:

```
if med eq 150 then
set local-preference 10
elseif med eq 200 then
set local-preference 60
elseif med eq 250 then
set local-preference 110
```
else
    set local-preference 0
endif

The statements within an if statement may themselves be if statements, as shown in the following example:

if community matches-any (12:34,56:78) then
    if med eq 150 then
        drop
    endif
    set local-preference 100
endif

This policy example sets the value of the local preference attribute to 100 on any route that has a community value of 12:34 or 56:78 associated with it. However, if any of these routes has a MED value of 150, then these routes with either the community value of 12:34 or 56:78 and a MED of 150 are dropped.

Note
Policy grammar allows user to enter simple if statements with optional else clauses on the same line. However, the grammar is restricted to single action or disposition statement. For detailed command options, enter match statement on a separate line.

Boolean Conditions

In the previous section describing the if statement, all of the examples use simple Boolean conditions that evaluate to either true or false. RPL also provides a way to build compound conditions from simple conditions by means of Boolean operators.

Three Boolean operators exist: negation (not), conjunction (and), and disjunction (or). In the policy language, negation has the highest precedence, followed by conjunction, and then by disjunction. Parentheses may be used to group compound conditions to override precedence or to improve readability.

The following simple condition:

med eq 42

is true only if the value of the MED in the route is 42, otherwise it is false.

A simple condition may also be negated using the not operator:

not next-hop in (10.0.2.2)

Any Boolean condition enclosed in parentheses is itself a Boolean condition:

(destination in prefix-list-1)

A compound condition takes either of two forms. It can be a simple expression followed by the and operator, itself followed by a simple condition:

med eq 42 and next-hop in (10.0.2.2)
A compound condition may also be a simpler expression followed by the or operator and then another simple condition:

\(\text{origin is igp or origin is incomplete}\)

An entire compound condition may be enclosed in parentheses:

\((\text{med eq 42 and next-hop in (10.0.2.2)})\)

The parentheses may serve to make the grouping of subconditions more readable, or they may force the evaluation of a subcondition as a unit.

In the following example, the highest-precedence not operator applies only to the destination test, the and operator combines the result of the not expression with the community test, and the or operator combines that result with the MED test.

\(\text{med eq 10 or not destination in (10.1.3.0/24) and community matches-any ([12..34]:[56..78])}\)

With a set of parentheses to express the precedence, the result is the following:

\(\text{med eq 10 or } (\text{not destination in (10.1.3.0/24)} \text{ and community matches-any ([12..34]:[56..78])})\)

The following is another example of a complex expression:

\((\text{origin is igp or origin is incomplete or not med eq 42}) \text{ and next-hop in (10.0.2.2)}\)

The left conjunction is a compound condition enclosed in parentheses. The first simple condition of the inner compound condition tests the value of the origin attribute; if it is Interior Gateway Protocol (IGP), then the inner compound condition is true. Otherwise, the evaluation moves on to test the value of the origin attribute again, and if it is incomplete, then the inner compound condition is true. Otherwise, the evaluation moves to check the next component condition, which is a negation of a simple condition.

apply

As discussed in the sections on policy definitions and parameterization of policies, the apply command executes another policy (either parameterized or unparameterized) from within another policy, which allows for the reuse of common blocks of policy. When combined with the ability to parameterize common blocks of policy, the apply command becomes a powerful tool for reducing repetitive configuration.

Attach Points

Policies do not become useful until they are applied to routes, and for policies to be applied to routes they need to be made known to routing protocols. In BGP, for example, there are several situations where policies can be used, the most common of these is defining import and export policy. The policy attach point is the point in which an association is formed between a specific protocol entity, in this case a BGP neighbor, and a specific named policy. It is important to note that a verification step happens at this point. Each time a policy is attached, the given policy and any policies it may apply are checked to ensure that the policy can be validly used at that attach point. For example, if a user defines a policy that sets the IS-IS level attribute and then
attempts to attach this policy as an inbound BGP policy, the attempt would be rejected because BGP routes do not carry IS-IS attributes. Likewise, when policies are modified that are in use, the attempt to modify the policy is verified against all current uses of the policy to ensure that the modification is compatible with the current uses.

Each protocol has a distinct definition of the set of attributes (commands) that compose a route. For example, BGP routes may have a community attribute, which is undefined in OSPF. Routes in IS-IS have a level attribute, which is unknown to BGP. Routes carried internally in the RIB may have a tag attribute.

When a policy is attached to a protocol, the protocol checks the policy to ensure the policy operates using route attributes known to the protocol. If the protocol uses unknown attributes, then the protocol rejects the attachment. For example, OSPF rejects attachment of a policy that tests the values of BGP communities.

The situation is made more complex by the fact that each protocol has access to at least two distinct route types. In addition to native protocol routes, for example BGP or IS-IS, some protocol policy attach points operate on RIB routes, which is the common central representation. Using BGP as an example, the protocol provides an attach point to apply policy to routes redistributed from the RIB to BGP. An attach point dealing with two different kinds of routes permits a mix of operations: RIB attribute operations for matching and BGP attribute operations for setting.

The protocol configuration rejects attempts to attach policies that perform unsupported operations.

The following sections describe the protocol attach points, including information on the attributes (commands) and operations that are valid for each attach point.

See Cisco IOS XR Routing Command Reference for the Cisco CRS Router for more information on the attributes and operations.

BGP Policy Attach Points

This section describes each of the BGP policy attach points and provides a summary of the BGP attributes and operators.

Additional-Path

The additional-path attach point provides increased control based on various attribute match operations. This attach point is used to decide whether a route-policy should be used to select additional-paths for a BGP speaker to be able to send multiple paths for the prefix.

The add path enables BGP prefix independent convergence (PIC) at the edge routers.

This example shows how to set a route-policy “add-path-policy” to be used for enabling selection of additional paths:

```
router bgp 100
  address-family ipv4 unicast
  additional-paths selection route-policy add-path-policy
```

Aggregation

The aggregation attach point generates an aggregate route to be advertised based on the conditional presence of subcomponents of that aggregate. Policies attached at this attach point are also able to set any of the valid
BGP attributes on the aggregated routes. For example, the policy could set a community value or a MED on the aggregate that is generated. The specified aggregate is generated if any routes evaluated by the named policy pass the policy. More specifics of the aggregate are filtered using the `suppress-route` keyword. Any actions taken to set attributes in the route affect attributes on the aggregate.

In the policy language, the configuration is controlled by which routes pass the policy. The suppress map was used to selectively filter or suppress specific components of the aggregate when the summary-only flag is not set. In other words, when the aggregate and more specific components are being sent, some of the more specific components can be filtered using a suppress map. In the policy language, this is controlled by selecting the route and setting the suppress flag. The attribute-map allowed the user to set specific attributes on the aggregated route. In the policy language, setting attributes on the aggregated route is controlled by normal action operations.

In the following example, the aggregate address 10.0.0.0/8 is generated if there are any component routes in the range 10.0.0.0/8 ge 8 le 25 except for 10.2.0.0/24. Because summary-only is not set, all components of the aggregate are advertised. However, the specific component 10.1.0.0 are suppressed.

```plaintext
route-policy sample
    if destination in (10.0.0.0/8 ge 8 le 25) then
        set community (10:33)
    endif
    if destination in (10.2.0.0/24) then
        drop
    endif
    if destination in (10.1.0.0/24) then
        suppress-route
    endif
end-policy

router bgp 2
    address-family ipv4
        aggregate-address 10.0.0.0/8 route-policy sample
```

The effect of aggregation policy on the attributes of the aggregate is cumulative. Every time an aggregation policy matches a more specific route, the set operations in the policy may modify the aggregate. The aggregate in the following example has a MED value that varies according to the number of more specific routes that comprise the aggregate.

```plaintext
route-policy bumping-aggregation
    set med +5
end-policy
```

If there are three matching more specific routes, the MED of the aggregate is the default plus 15; if there are seventeen more specific routes, the MED of the aggregate is the default plus 85.

The order that the aggregation policy is applied to prefix paths is deterministic but unspecified. That is, a given set of routes always appears in the same order, but there is no way to predict the order.

A drop in aggregation policy does not prevent generation of an aggregate, but it does prevent the current more specific route from contributing to the aggregate. If another more specific route gives the route a pass, the aggregate is generated. Only one more specific pass is required to generate an aggregate.
Dampening

The dampening attach point controls the default route-dampening behavior within BGP. Unless overridden by a more specific policy on the associate peer, all routes in BGP apply the associated policy to set their dampening attributes.

The following policy sets dampening values for BGP IPv4 unicast routes. Those routes that are more specific than a /25 take longer to recover after they have been dampened than routes that are less specific than /25.

```plaintext
route-policy sample_damp
  if destination in (0.0.0.0/0 ge 25) then
    set dampening halflife 30 others default
  else
    set dampening halflife 20 others default
  endif
end-policy
```

```plaintext
router bgp 2
  address-family ipv4 unicast
    bgp dampening route-policy sample_damp
```

Default Originate

The default originate attach point allows the default route (0.0.0.0/0) to be conditionally generated and advertised to a peer, based on the presence of other routes. It accomplishes this configuration by evaluating the associated policy against routes in the Routing Information Base (RIB). If any routes pass the policy, the default route is generated and sent to the relevant peer.

The following policy generates and sends a default-route to the BGP neighbor 10.0.0.1 if any routes that match 10.0.0.0/8 ge 8 le 32 are present in the RIB.

```plaintext
route-policy sample-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 32) then
    pass
  endif
end-policy
```

```plaintext
router bgp 2
  neighbor 10.0.0.1
    remote-as 3
    address-family ipv4 unicast
      default-originate policy sample-originate
```

Neighbor Export

The neighbor export attach point selects the BGP routes to send to a given peer or group of peers. The routes are selected by running the set of possible BGP routes through the associated policy. Any routes that pass the policy are then sent as updates to the peer or group of peers. The routes that are sent may have had their BGP attributes altered by the policy that has been applied.
The following policy sends all BGP routes to neighbor 10.0.0.5. Routes that are tagged with any community in the range 2:100 to 2:200 are sent with a MED of 100 and a community of 2:666. The rest of the routes are sent with a MED of 200 and a community of 2:200.

```plaintext
route-policy sample-export
    if community matches-any (2:[100-200]) then
        set med 100
        set community (2:666)
    else
        set med 200
        set community (2:200)
    endif
end-policy

router bgp 2
    neighbor 10.0.0.5
        remote-as 3
    address-family ipv4 unicast
        route-policy sample-export out
```

**Neighbor Import**

The neighbor import attach point controls the reception of routes from a specific peer. All routes that are received by a peer are run through the attached policy. Any routes that pass the attached policy are passed to the BGP Routing Information Base (BRIB) as possible candidates for selection as best path routes.

When a BGP import policy is modified, it is necessary to rerun all the routes that have been received from that peer against the new policy. The modified policy may now discard routes that were previously allowed through, allow through previously discarded routes, or change the way the routes are modified. A new configuration option in BGP (**bgp auto-policy-soft-reset**) that allows this modification to happen automatically in cases for which either soft reconfiguration is configured or the BGP route-refresh capability has been negotiated.

The following example shows how to receive routes from neighbor 10.0.0.1. Any routes received with the community 3:100 have their local preference set to 100 and their community tag set to 2:666. All other routes received from this peer have their local preference set to 200 and their community tag set to 2:200.

```plaintext
route-policy sample_import
    if community matches-any (3:100) then
        set local-preference 100
        set community (2:666)
    else
        set local-preference 200
        set community (2:200)
    endif
end-policy

router bgp 2
    neighbor 10.0.0.1
        remote-as 3
    address-family ipv4 unicast
        route-policy sample_import in
```

---

*Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x*
Network

The network attach point controls the injection of routes from the RIB into BGP. A route policy attached at this point is able to set any of the valid BGP attributes on the routes that are being injected.

The following example shows a route policy attached at the network attach point that sets the well-known community no-export for any routes more specific than /24:

```plaintext
route-policy NetworkControl
  if destination in (0.0.0.0/0 ge 25) then
    set community (no-export) additive
  endif
end-policy
router bgp 2
  address-family ipv4 unicast
    network 172.16.0.5/27 route-policy NetworkControl
```

Redistribute

The redistribute attach point allows routes from other sources to be advertised by BGP. The policy attached at this point is able to set any of the valid BGP attributes on the routes that are being redistributed. Likewise, selection operators allow a user to control what route sources are being redistributed and which routes from those sources.

The following example shows how to redistribute all routes from OSPF instance 12 into BGP. If OSPF were carrying a default route, it is dropped. Routes carrying a tag of 10 have their local preference set to 300 and the community value of 2:666 and no-advertise attached. All other routes have their local preference set to 200 and a community value of 2:100 set.

```plaintext
route-policy sample_redistribute
  if destination in (0.0.0.0/0) then
    drop
  endif
  if tag eq 10 then
    set local-preference 300
    set community (2:666, no-advertise)
  else
    set local-preference 200
    set community (2:100)
  endif
end-policy
router bgp 2
  address-family ipv4 unicast
    redistribute ospf 12 route-policy sample_redistribute
```

Show BGP

The show bgp attach point allows the user to display selected BGP routes that pass the given policy. Any routes that are not dropped by the attached policy are displayed in a manner similar to the output of the `show bgp` command.
In the following example, the `show bgp route-policy` command is used to display any BGP routes carrying a MED of 5:

```
route-policy sample-display
  if med eq 5 then
    pass
  endif
end-policy
!
show bgp route-policy sample-display
```

A `show bgp policy route-policy` command also exists, which runs all routes in the RIB past the named policy as if the RIB were an outbound BGP policy. This command then displays what each route looked like before it was modified and after it was modified, as shown in the following example:

```
show rpl route-policy test2
route-policy test2
  if (destination in (10.0.0.0/8 ge 8 le 32)) then
    set med 333
  endif
end-policy
!
show bgp
```

```
BGP router identifier 10.0.0.1, local AS number 2
BGP main routing table version 11
BGP scan interval 60 secs
Status codes: suppressed, d damped, h history, * valid, > best
  i - internal, S stale
Origin codes: i - IGP, e - EGP, ? - incomplete

Network      Next Hop      Metric LocPrf Weight Path
*> 10.0.0.0   10.0.1.2     10    0  3  ?
*> 10.0.0.0/9 10.0.1.2     10    0  3  ?
*> 10.0.0.0/10 10.0.1.2    10    0  3  ?
*> 10.0.0.0/11 10.0.1.2    10    0  3  ?
*> 10.1.0.0/16 10.0.1.2    10    0  3  ?
*> 10.3.30.0/24 10.0.1.2   10    0  3  ?
*> 10.3.30.128/25 10.0.1.2 10    0  3  ?
*> 10.128.0.0/9 10.0.1.2   10    0  3  ?
*> 10.255.0.0/24 10.0.101.2 1000 555  0  100  e
*> 10.255.64.0/24 10.0.101.2 1000 555  0  100  e
....
```

```
show bgp policy route-policy test2
10.0.0.0/8 is advertised to 10.0.101.2
Path info:
  neighbor:10.0.1.2   neighbor router id:10.0.1.2
    valid external best
Attributes after inbound policy was applied:
  next hop:10.0.1.2
  MET ORG AS
  origin:incomplete neighbor as:3 metric:10
  aspath:3
Attributes after outbound policy was applied:
  next hop:10.0.1.2
  MET ORG AS
  origin:incomplete neighbor as:3 metric:333
  aspath:2 3
...
**Table Policy**

The table policy attach point allows the user to configure traffic-index values on routes as they are installed into the global routing table. This attach point supports the BGP policy accounting feature. BGP policy accounting uses the traffic indexes that are set on the BGP routes to track various counters. This way, router operators can select different sets of BGP route attributes using the matching operations and then set different traffic indexes for each different class of route they are interested in tracking.

The following example shows how to set the traffic index to 10 in IPv4 unicast routes that originated from autonomous system 10.33. Likewise, any IPv4 unicast routes that originated from autonomous system 11.60 have their traffic index set to 11 when they are installed into the FIB. These traffic indexes are then used to count traffic being forwarded on these routes inline cards by enabling the BGP policy accounting counters on the interfaces of interest.

```
route-policy sample-table
   if as-path originates-from '10.33' then
      set traffic-index 10
   elseif as-path originates-from '11.60' then
      set traffic-index 11
   endif
end-policy

router bgp 2
   address-family ipv4 unicast
      table-policy sample-table
```

**Import**

The import attach point provides control over the import of routes from the global VPN IPv4 table to a particular VPN routing and forwarding (VRF) instance.

For Layer 3 VPN networks, provider edge (PE) routers learn of VPN IPv4 routes through the Multiprotocol Internal Border Gateway Protocol (MP-iBGP) from other PE routers and automatically filters out route announcements that do not contain route targets that match any import route targets of its VRFs.

This automatic route filtering happens without RPL configuration; however, to provide more control over the import of routes in a VRF, you can configure a VRF import policy.

The following example shows how to perform matches based on a route target extended community and then sets the next hop. If the route has route target value 10:91, then the next hop is set to 172.16.0.1. If the route has route target value 11:92, then the next hop is set to 172.16.0.2. If the route has Site-of-Origin (SoO) value 10:111111 or 10:111222, then the route is dropped. All other non-matching routes are dropped.

```
route-policy bgpvrf_import
   if extcommunity rt matches-any (10:91) then
      set next-hop 172.16.0.1
   elseif extcommunity rt matches-every (11:92) then
      set next-hop 172.16.0.2
   elseif extcommunity soo matches-any (10:111111, 10:111222) then
      pass
   endif
end-policy

vrf vrf_import
   address-family ipv4 unicast
      import route-policy bgpvrf_import
```

Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x
Note

'Set' is a valid operator for the 'med' attribute at the bgp import attach point.

---

Export

The export attach point provides control over the export of routes from a particular VRF to a global VPN IPv4 table.

For Layer 3 VPN networks, export route targets are added to the VPN IPv4 routes when VRF IPv4 routes are converted into VPN IPv4 routes and advertised through the MP-iBGP to other PE routers (or flow from one VRF to another within a PE router).

A set of export route targets is configured with the VRF without RPL configuration; however, to set route targets conditionally, you can configure a VRF export policy.

The following example shows some match and set operations supported for the export route policy. If a route matches 172.16.1.0/24 then the route target extended community is set to 10:101, and the weight is set to 211. If the route does not match 172.16.1.0/24 but the origin of the route is egp, then the local preference is set to 212 and the route target extended community is set to 10:101. If the route does not match those specified criteria, then the route target extended community 10:111222 is added to the route. In addition, RT 10:111222 is added to the route that matches any of the previous conditions as well.

```plaintext
route-policy bgpvrf_export
  if destination in (172.16.1.0/24) then
    set extcommunity rt (10:101)
    set weight 211
  elseif origin is egp then
    set local-preference 212
    set extcommunity rt (10:101)
  endif
  set extcommunity rt (10:111222) additive
end-policy

vrf vrf-export
  address-family ipv4 unicast
    export route-policy bgpvrf-export
      .
      .
```

---

Note

'Set' is a valid operator for the 'med' attribute at the bgp export attach point.

---

Retain Route-Target

The retain route target attach point within BGP allows the specification of match criteria based only on route target extended community. The attach point is useful at the route reflector (RR) or at the Autonomous System Boundary Router (ASBR).

Typically, an RR has to retain all IPv4 VPN routes to peer with its PE routers. These PEs might require routers tagged with different route target IPv4 VPN routes resulting in non-scaleable RRs. You can achieve scalability
if you configure an RR to retain routes with a defined set of route target extended communities, and a specific set of VPNs to service.

Another reason to use this attach point is for an ASBR. ASBRs do not require that VRFs be configured, but need this configuration to retain the IPv4 VPN prefix information.

The following example shows how to configure the route policy retainer and apply it to the retain route target attach point. The route is accepted if the route contains route target extended communities 10:615, 10:6150, and 15.15.15.15:15. All other non-matching routes are dropped.

```plaintext
extcommunity-set rt rtset1  
0:615,  
10:6150,  
15.15.15.15:15  
end-set  

route-policy retainer  
  if extcommunity rt matches-any rtset1 then  
    pass  
  endif  
end-policy  

router bgp 2  
  address-family vpnv4 unicast  
    retain route-target route-policy retainer  
  .  
  .
```

**Allocate-Label**

The allocate-label attach point provides increased control based on various attribute match operations. This attach point is typically used in inter-AS option C to decide whether the label should be allocated or not when sending updates to the neighbor for the IPv4 labeled unicast address family. The attribute setting actions supported are for pass and drop.

The following example shows how to configure a route policy that passes the prefix 0.0.0.0 with prefix length 0. Label allocation happens only if prefix 0.0.0.0 exists.

```plaintext
route-policy label_policy  
  if destination in (0.0.0.0/0) then  
    pass  
  endif  
end-policy  

router bgp 2  
  vrf vrf1  
    rd auto  
    address-family ipv4 unicast  
      allocate-label route-policy label-policy  
  .  
  .
```

**Neighbor-ORF**

The neighbor-orf attach point provides the filtering of incoming BGP route updates using only prefix-based matching. In addition to using this as an inbound filter, the prefixes and disposition (drop or pass) are sent to upstream neighbors as an Outbound Route Filter (ORF) to allow them to perform filtering.
The following example shows how to configure a route policy orf-preset and apply it to the neighbor ORF attach point. The prefix of the route is dropped if it matches any prefix specified in orf-preset (172.16.1.0/24, 172.16.5.0/24, 172.16.11.0/24). In addition to this inbound filtering, BGP also sends these prefix entries to the upstream neighbor with a permit or deny so that the neighbor can filter updates before sending them on to their destination.

```plaintext
prefix-set orf-preset
  172.16.1.0/24,
  172.16.5.0/24,
  172.16.11.0/24
end-set

route-policy policy-orf
  if orf prefix in orf-preset then
    drop
  endif
  if orf prefix in (172.16.3.0/24, 172.16.7.0/24, 172.16.13.0/24) then
    pass
  endif

router bgp 2
  neighbor 1.1.1.1
    remote-as 3
    address-family ipv4 unicast
      orf route-policy policy-orf
```

Next-hop

The next-hop attach point provides increased control based on protocol and prefix-based match operations. The attach point is typically used to decide whether to act on a next-hop notification (up or down) event.

Support for next-hop tracking allows BGP to monitor reachability for routes in the Routing Information Base (RIB) that can directly affect BGP prefixes. The route policy at the BGP next-hop attach point helps limit notifications delivered to BGP for specific prefixes. The route policy is applied on RIB routes. Typically, route policies are used in conjunction with next-hop tracking to monitor non-BGP routes.

The following example shows how to configure the BGP next-hop tracking feature using a route policy to monitor static or connected routes with the prefix 10.0.0.0 and prefix length 8.

```plaintext
route-policy nxthp_policy_A
  if destination in (10.0.0.0/8) and protocol in (static, connected) then
    pass
  endif
end-policy

router bgp 2
  address-family ipv4 unicast
    nexthop route-policy nxthp_policy_A
```

Clear-Policy

The clear-policy attach point provides increased control based on various AS path match operations when using a `clear bgp` command. This attach point is typically used to decide whether to clear BGP flap statistics based on AS-path-based match operations.
The following example shows how to configure a route policy where the in operator evaluates to true if one or more of the regular expression matches in the set my-as-set successfully match the AS path associated with the route. If it is a match, then the clear command clears the associated flap statistics.

```
as-path-set my-as-set
  ios-regex '_12$'
  ios-regex '_13$'
end-set

route-policy policy_a
  if as-path in my-as-set then
    pass
  else
    drop
  endif
end-policy

clear bgp ipv4 unicast flap-statistics route-policy policy_a
```

**Debug**

The debug attach point provides increased control based on prefix-based match operations. This attach point is typically used to filter debug output for various BGP commands based on the prefix of the route.

The following example shows how to configure a route policy that will only pass the prefix 20.0.0.0 with prefix length 8; therefore, the debug output shows up only for that prefix.

```
route-policy policy_b
  if destination in (10.0.0.0/8) then
    pass
  else
    drop
  endif
end-policy

d debug bgp update policy_b
```

**BGP Attributes and Operators**

This table summarizes the BGP attributes and operators per attach points.

*Table 1: BGP Attributes and Operators*

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional-paths</td>
<td>path-selection</td>
<td>—</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-empty</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>aggregation</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td>as-path-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>as-path-unique-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>community</td>
<td>is-empty</td>
<td>set</td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td>matches-any</td>
<td>delete in</td>
<td>delete not in</td>
</tr>
<tr>
<td></td>
<td>matches-every</td>
<td>delete all</td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td>in</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>extcommunity cost</td>
<td>n/a</td>
<td>set</td>
<td>set additive</td>
</tr>
<tr>
<td>local-preference</td>
<td>is, eg, ge, le</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>med</td>
<td>is, eg, ge, le</td>
<td>set</td>
<td>set+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set-</td>
<td></td>
</tr>
<tr>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>origin</td>
<td>is</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>in</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>suppress-route</td>
<td>n/a</td>
<td>suppress-route</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>n/a</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>--------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>allocate-label</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td>as-path-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>as-path-unique-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>label</td>
<td></td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>local-preference</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>med</td>
<td></td>
<td>is, eg, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td>next-hop</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>origin</td>
<td></td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td>source</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>clear-policy</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>dampening</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is originates-from passes-through unique-length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as-path-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>as-path-unique-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>is-empty matches-any matches-every</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>dampening</td>
<td>n/a</td>
<td>set dampening</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To set values that control the dampening (see Dampening, on page 29)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>local-preference</td>
<td>is, eg, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>is, eg, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>origin</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>debug</td>
<td>destination</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>default originate</td>
<td>as-path</td>
<td>n/a</td>
<td>prepend</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>community with 'peeras'</td>
<td>n/a</td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td>extcommunity cost</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>extcommunity rt</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>extcommunity soo</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>local-preference</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>origin</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-has-route</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>----------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>export</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td>as-path-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>as-path-unique-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td>is-empty</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>extcommunity rt</td>
<td></td>
<td>is-empty</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td>delete-in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td>delete-not-in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete-all</td>
</tr>
<tr>
<td>extcommunity soo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local-preference</td>
<td></td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>med</td>
<td></td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>next-hop</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>origin</td>
<td></td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td>source</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>--------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>import</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through unique-length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as-path-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>as-path-unique-length</td>
<td>is, ge, le, eq</td>
<td></td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>extcommunity rt</td>
<td>extcommunity rt</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td>extcommunity soo</td>
<td>extcommunity soo</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td>local-preference</td>
<td>local-preference</td>
<td>is, ge, le, eq</td>
<td>set</td>
</tr>
<tr>
<td>med</td>
<td>med</td>
<td>is, eg, ge, le</td>
<td>set</td>
</tr>
<tr>
<td>next-hop</td>
<td>next-hop</td>
<td>in</td>
<td>set peer address set destination vrf</td>
</tr>
<tr>
<td>origin</td>
<td>origin</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td>source</td>
<td>source</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>weight</td>
<td>weight</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>--------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>neighbor-in</td>
<td>as-path</td>
<td>in</td>
<td>prepend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td>prepend most-recent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td>replace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as-path-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>as-path-unique-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>communitycommunity with 'peeras'</td>
<td>is-empty</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td>delete in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete not in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete all</td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>extcommunity cost</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td>extcommunity rt</td>
<td>is-empty</td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td>delete-in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td>delete-not-in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td>delete-all</td>
</tr>
<tr>
<td></td>
<td>extcommunity soo</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td></td>
<td>local-preference</td>
<td>is, ge, le, eq</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>is, eg, ge, le</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set-</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td></td>
</tr>
</tbody>
</table>
### Attach Points

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set peer address</td>
</tr>
<tr>
<td>origin</td>
<td>is</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>path-type</td>
<td>is</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>source</td>
<td>in</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>vpn-distinguisher</td>
<td>is</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>weight</td>
<td>n/a</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------</td>
<td>---------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>neighbor-out</td>
<td>as-path</td>
<td>in</td>
<td>prepend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td>prepend most-recent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td>replace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td>as-path-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>as-path-unique-length</td>
<td></td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td>is-empty</td>
<td>set</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td>matches-any</td>
<td>set additive</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td>matches-every</td>
<td>delete in</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td></td>
<td>delete not in</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td></td>
<td>delete all</td>
</tr>
<tr>
<td>community</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>community</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td></td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extcommunity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extcommunity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extcommunity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local-preference</td>
<td></td>
<td>is, eg, ge, le</td>
<td>set</td>
</tr>
<tr>
<td>med</td>
<td></td>
<td>is, eg, ge, le</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set-</td>
</tr>
<tr>
<td>next-hop</td>
<td></td>
<td>in</td>
<td></td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set self</td>
</tr>
<tr>
<td>origin</td>
<td>is</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>path-type</td>
<td>is</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>rd</td>
<td>in</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>source</td>
<td>in</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>unsuppress-route</td>
<td>n/a</td>
<td></td>
<td>unsuppress-route</td>
</tr>
<tr>
<td>vpn-distinguisher</td>
<td>n/a</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>weight</td>
<td>is</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>neighbor-orf</td>
<td>orf-prefix</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>network</td>
<td>as-path</td>
<td>n/a</td>
<td>prepend</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete not in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete all</td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>extcommunity cost</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td>local-preference</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set-</td>
</tr>
<tr>
<td>next-hop</td>
<td>n/a</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>origin</td>
<td>n/a</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>route-type</td>
<td>is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tag</td>
<td>is, eg, ge, le</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>n/a</td>
<td></td>
<td>set</td>
</tr>
<tr>
<td>next-hop</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>protocol</td>
<td>is, in</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>Attach Point</td>
<td>Attribute</td>
<td>Match</td>
<td>Set</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td>redistribute</td>
<td>as-path</td>
<td>n/a</td>
<td>prepend</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete not in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete all</td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>extcommunity cost</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set additive</td>
</tr>
<tr>
<td></td>
<td>local-preference</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set-</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>origin</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>is, eq, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-has-label</td>
<td>route-has-label</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-type</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>retain-rt</td>
<td>extcommunity rt</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
</tbody>
</table>

Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x
<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td>as-path</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>neighbor-is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>originates-from</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>passes-through</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique-length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as-path-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>as-path-unique-length</td>
<td>is, ge, le, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>extcommunity rt</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extcommunity soo</td>
<td>is-empty</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-any</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-every</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches-within</td>
<td></td>
</tr>
<tr>
<td></td>
<td>med</td>
<td>is, eg, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>origin</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>in</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Some BGP route attributes are inaccessible from some BGP attach points for various reasons. For example, the `set med igp-cost only` command makes sense when there is a configured igp-cost to provide a source value.

This table summarizes which operations are valid and where they are valid.

**Table 2: Restricted BGP Operations by Attach Point**

<table>
<thead>
<tr>
<th>Command</th>
<th>import</th>
<th>export</th>
<th>aggregation</th>
<th>redistribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepend as-path most-recent</td>
<td>eBGP only</td>
<td>eBGP only</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
RPL - if prefix is-best-path/is-best-multipath

Border Gateway Protocol (BGP) routers receive multiple paths to the same destination. As a standard, by default the BGP best path algorithm decides the best path to install in IP routing table. This is used for traffic forwarding.

BGP assigns the first valid path as the current best path. It then compares the best path with the next in the list. This process continues, until BGP reaches the end of the list of valid paths. This contains all rules used to determine the best path. When there are multiple paths for a given address prefix, BGP:

- Selects one of the paths as the best path as per the best-path selection rules.
- Installs the best path in its forwarding table. Each BGP speaker advertises only the best-path to its peers.

The advertisement rule of sending only the best path does not convey the full routing state of a destination, present on a BGP speaker to its peers.

After the BGP speaker receives a path from one of its peers; the path is used by the peer for forwarding packets. All other peers receive the same path from this peer. This leads to a consistent routing in a BGP network. To improve the link bandwidth utilization, most BGP implementations choose additional paths satisfy certain conditions, as multi-path, and install them in the forwarding table. Incoming packets for such are load-balanced across the best-path and the multi-path(s). You can install the paths in the forwarding table that are not advertised to the peers. The RR route reflector finds out the best-path and multi-path. This way the route reflector uses different communities for best-path and multi-path. This feature allows BGP to signal the local decision done by RR or Border Router. With this new feature, selected by RR using community-string (if is-best-path then community 100:100). The controller checks which best path is sent to all R's. Border Gateway Protocol routers receive multiple paths to the same destination. While carrying out best path computation there will be one best path, sometimes equal and few non-equal paths. Thus, the requirement for a best-path and is-equal-best-path.

The BGP best path algorithm decides the best path in the IP routing table and used for forwarding traffic. This enhancement within the RPL allows creating policy to take decisions. Adding community-string for local selection of best path. With introduction of BGP Additional Path (Add Path), BGP now signals more than the best Path. BGP can signal the best path and the entire path equivalent to the best path. This is in accordance to the BGP multi-path rules and all backup paths.

---

**Command** | **import** | **export** | **aggregation** | **redistribution**
--- | --- | --- | --- | ---
replace as-path | eBGP only | eBGP only | n/a | n/a
set med igp-cost | forbidden | eBGP only | forbidden | forbidden
set weight | n/a | forbidden | n/a | n/a
suppress | forbidden | forbidden | n/a | forbidden
**OSPF Policy Attach Points**

This section describes each of the OSPF policy attach points and provides a summary of the OSPF attributes and operators.

**Default-Information Originate**

The default-information originate attach point allows the user to conditionally inject the default route 0.0.0.0/0 into the OSPF link-state database, which is done by evaluating the attached policy. If any routes in the local RIB pass the policy, then the default route is inserted into the link-state database.

The following example shows how to generate a default route if any of the routes that match 10.0.0.0/8 ge 8 le 25 are present in the RIB:

```plaintext
route-policy ospf-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 25) then
    pass
  endif
end-policy
router ospf 1
  default-information originate policy ospf-originate

```

**Redistribute**

The redistribute attach point within OSPF injects routes from other routing protocol sources into the OSPF link-state database, which is done by selecting the routes it wants to import from each protocol. It then sets the OSPF parameters of cost and metric type. The policy can control how the routes are injected into OSPF by using the `set metric-type` or `set ospf-metric` command.

The following example shows how to redistribute routes from IS-IS instance instance_10 into OSPF instance 1 using the policy OSPF-redist. The policy sets the metric type to type-2 for all redistributed routes. IS-IS routes with a tag of 10 have their cost set to 100, and IS-IS routes with a tag of 20 have their OSPF cost set to 200. Any IS-IS routes not carrying a tag of either 10 or 20 are not be redistributed into the OSPF link-state database.

```plaintext
route-policy OSPF-redist
  set metric-type type-2
  if tag eq 10 then
    set ospf cost 100
  elseif tag eq 20 then
    set ospf cost 200
  else
    drop
  endif
end-policy
router ospf 1
  redistribute isis instance_10 policy OSPF-redist

```
**Area-in**

The area-in attach point within OSPF allows you to filter inbound OSPF type-3 summary link-state advertisements (LSAs). The attach point provides prefix-based matching and hence increased control for filtering type-3 summary LSAs.

The following example shows how to configure the prefix for OSPF summary LSAs. If the prefix matches any of 111.105.3.0/24, 111.105.7.0/24, 111.105.13.0/24, it is accepted. If the prefix matches any of 111.106.3.0/24, 111.106.7.0/24, 111.106.13.0/24, it is dropped.

```
route-policy OSPF-area-in
  if destination in 
    111.105.3.0/24,
    111.105.7.0/24,
    111.105.13.0/24) then
      drop
  endif
  if destination in 
    111.106.3.0/24,
    111.106.7.0/24,
    111.106.13.0/24) then
    pass
  endif
end-policy
router ospf 1
  area 1
    route-policy OSPF-area-in in
```

**Area-out**

The area-out attach point within OSPF allows you to filter outbound OSPF type-3 summary LSAs. The attach point provides prefix-based matching and, hence, increased control for filtering type-3 summary LSAs.

The following example shows how to configure the prefix for OSPF summary LSAs. If the prefix matches any of 211.105.3.0/24, 211.105.7.0/24, 211.105.13.0/24, it is announced. If the prefix matches any of 212.105.3.0/24, 212.105.7.0/24, 212.105.13.0/24, it is dropped and not announced.

```
route-policy OSPF-area-out
  if destination in 
    211.105.3.0/24,
    211.105.7.0/24,
    211.105.13.0/24) then
      drop
  endif
  if destination in 
    212.105.3.0/24,
    212.105.7.0/24,
    212.105.13.0/24) then
    pass
  endif
end-policy
router ospf 1
  area 1
    route-policy OSPF-area-out out
```
**SPF Prefix-priority**

The spf-prefix-priority attach point within OSPF allows you to define the route policy to apply to OSPFv2 prefix prioritization.

**OSPF Attributes and Operators**

This table summarizes the OSPF attributes and operators per attach points.

*Table 3: OSPF Attributes and Operators*

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>distribute-list-in-area</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>eq, ge, is, le</td>
<td>n/a</td>
</tr>
<tr>
<td>distribute-list-in-instance</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>eq, ge, is, le</td>
<td>n/a</td>
</tr>
<tr>
<td>distribute-list-in-interface</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>eq, ge, is, le</td>
<td>n/a</td>
</tr>
<tr>
<td>default-information originate</td>
<td>ospf-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>metric-type</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-has-route</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>redistribute</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>metric-type</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>ospf-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>is, le, ge, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-has-level</td>
<td>route-has-level</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-type</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, le, ge, le</td>
<td>set</td>
</tr>
</tbody>
</table>
Distribute-list in

The distribute-list in attach point within OSPF allows use of route policies to filter OSPF prefixes. The distribute-list in route-policy can be configured at OSPF instance, area, and interface levels. The route-policy used in the distribute-list in command supports match statements, "destination" and "rib-metric". The "set" commands are not supported in the route-policy.

These are examples of valid route-policies for "distribute-list in":

```plaintext
route-policy DEST
  if destination in (10.10.10.10/32) then
    drop
  else
    pass
  endif
end-policy

route-policy METRIC
  if rib-metric ge 10 and rib-metric le 19 then
    drop
  else
    pass
  endif
end-policy

prefix-set R-PFX
10.10.10.30
end-set

route-policy R-SET
  if destination in R-PFX and rib-metric le 20 then
    pass
  else
    drop
  endif
end-policy
```

### OSPFv3 Policy Attach Points

This section describes each of the OSPFv3 policy attach points and provides a summary of the OSPFv3 attributes and operators.

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>area-in</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>area-out</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>spf-prefix-priority</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>spf-priority</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, le, ge, eq</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Default-Information Originate

The default-information originate attach point allows the user to conditionally inject the default route 0::/0 into the OSPFv3 link-state database, which is done by evaluating the attached policy. If any routes in the local RIB pass the policy, then the default route is inserted into the link-state database.

The following example shows how to generate a default route if any of the routes that match 2001::/96 are present in the RIB:

```
route-policy ospfv3-originate
  if rib-has-route in (2001::/96) then
    pass
  endif
end-policy

router ospfv3 1
  default-information originate policy ospfv3-originate
```

Redistribute

The redistribute attach point within OSPFv3 injects routes from other routing protocol sources into the OSPFv3 link-state database, which is done by selecting the route types it wants to import from each protocol. It then sets the OSPFv3 parameters of cost and metric type. The policy can control how the routes are injected into OSPFv3 by using the metric type command.

The following example shows how to redistribute routes from BGP instance 15 into OSPF instance 1 using the policy OSPFv3-redist. The policy sets the metric type to type-2 for all redistributed routes. BGP routes with a tag of 10 have their cost set to 100, and BGP routes with a tag of 20 have their OSPFv3 cost set to 200. Any BGP routes not carrying a tag of either 10 or 20 are not be redistributed into the OSPFv3 link-state database.

```
route-policy OSPFv3-redist
  set metric-type type-2
  if tag eq 10 then
    set extcommunity cost 100
  elseif tag eq 20 then
    set extcommunity cost 200
  else
    drop
  endif
end-policy

router ospfv3 1
  redistribute bgp 15 policy OSPFv3-redist
```

OSPFv3 Attributes and Operators

This table summarizes the OSPFv3 attributes and operators per attach points.
Table 4: OSPFv3 Attributes and Operators

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-information</td>
<td>ospf-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>originate</td>
<td>metric-type</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-has-route</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>redistribute</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>metric-type</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>ospf-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>is, le, ge, eq</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-has-level</td>
<td>route-has-level</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-type</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, le, ge, eq</td>
<td>set</td>
</tr>
</tbody>
</table>

IS-IS Policy Attach Points

This section describes each of the IS-IS policy attach points and provides a summary of the IS-IS attributes and operators.

Redistribute

The redistribute attach point within IS-IS allows routes from other protocols to be readvertised by IS-IS. The policy is a set of control structures for selecting the types of routes that a user wants to redistribute into IS-IS. The policy can also control which IS-IS level the routes are injected into and at what metric values.

The following describes an example. Here, routes from IS-IS instance 1 are redistributed into IS-IS instance instance_10 using the policy ISIS-redist. This policy sets the level to level-1-2 for all redistributed routes. IS-IS routes with a tag of 10 have their metric set to 100, and IS-IS routes with a tag of 20 have their IS-IS metric set to 200. Any IS-IS routes not carrying a tag of either 10 or 20 are not be redistributed into the IS-IS database.

```
route-policy ISIS-redist
set level level-1-2
if tag eq 10 then
  set isis-metric 100
elseif tag eq 20 then
  set isis-metric 200
```
else
    drop
endif
end-policy

router isis instance_10
    address-family ipv4 unicast
        redistribute isis 1 policy ISIS-redist
        .
        .

Default-Information Originate

The default-information originate attach point within IS-IS allows the default route 0.0.0.0/0 to be conditionally injected into the IS-IS route database.

The following example shows how to generate an IPv4 unicast default route if any of the routes that match 10.0.0.0/8 ge 8 le 25 is present in the RIB. The cost of the IS-IS route is set to 100 and the level is set to level-1-2 on the default route that is injected into the IS-IS database.

route-policy isis-originate
    if rib-has-route in (10.0.0.0/8 ge 8 le 25) then
        set metric 100
        set level level-1-2
    endif
end-policy

router isis instance_10
    address-family ipv4 unicast
        default-information originate policy isis_originate
        .

Inter-area-propagate

The inter-area-propagate attach point within IS-IS allows the prefixes to be conditionally propagated from one level to another level within the same IS-IS instance.

The following example shows how to allow prefixes to be leaked from the level 1 LSP into the level 2 LSP if any of the prefixes match 10.0.0.0/8 ge 8 le 25.

route-policy isis-propagate
    if destination in (10.0.0.0/8 ge 8 le 25) then
        pass
    endif
end-policy

router isis instance_10
    address-family ipv4 unicast
        propagate level 1 into level 2 policy isis-propagate
        .

IS-IS Attributes and Operators

This table summarizes the IS-IS attributes and operators per attach points.
Table 5: IS-IS Attributes and Operators

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>redistribution</td>
<td>tag</td>
<td>is, le, ge</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-type</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following route-type cannot be matched: ospf-nssa-type-1 and ospf-nssa-type-2</td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td>in</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>mpls-label</td>
<td>route-has-label</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>rib-metric</td>
<td>is, le, ge, eq</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>level</td>
<td>n/a</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>isis-metric</td>
<td>n/a</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td>n/a</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>metric-type</td>
<td>n/a</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>default-information</td>
<td>rib-has-route</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>originate</td>
<td>level</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>isis-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td>inter-area-propagate</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**EIGRP Policy Attach Points**

This section describes each of the EIGRP policy attach points and provides a summary of the EIGRP attributes and operators.
Default-Accept-In

The default-accept-in attach point allows you to set and reset the conditional default flag for EIGRP routes by evaluating the attached policy.

The following example shows a policy that sets the conditional default flag for all routes that match 10.0.0.0/8 and longer prefixes up to 10.0.0.0/25:

```
route-policy eigrp-cd-policy-in
  if destination in (10.0.0.0/8 ge 8 le 25) then
    pass
  endif
end-policy
```

```
router eigrp 100
  address-family ipv4
    default-information allowed in route-policy eigrp-cd-policy-in
    
    
```

Default-Accept-Out

The default-accept-out attach point allows you to set and reset the conditional default flag for EIGRP routes by evaluating the attached policy.

The following example shows a policy that sets the conditional default flag for all routes that match 100.10.0.0/16:

```
route-policy eigrp-cd-policy-out
  if destination in (200.10.0.0/16) then
    pass
  endif
end-policy
```

```
router eigrp 100
  address-family ipv4
    default-information allowed out route-policy eigrp-cd-policy-out
    
    
```

Policy-In

The policy-in attach point allows you to filter and modify inbound EIGRP routes. This policy is applied to all interfaces for which there is no interface inbound route policy.

The following example shows the command under EIGRP:

```
router eigrp 100
  address-family ipv4
    route-policy global-policy-in in
    
    
```
Policy-Out

The policy-out attach point allows you to filter and modify outbound EIGRP routes. This policy is applied to all interfaces for which there is no interface outbound route policy.

The following example shows the command under EIGRP:

```
router eigrp 100
    address-family ipv4
    route-policy global-policy-out out
```

If-Policy-In

The if-policy-in attach point allows you to filter routes received on a particular EIGRP interface. The following example shows an inbound policy for GigabitEthernet interface 0/2/0/3:

```
router eigrp 100
    address-family ipv4
    interface GigabitEthernet0/2/0/3
    route-policy if-filter-policy-in in
```

If-Policy-Out

The if-policy-out attach point allows you to filter routes sent out on a particular EIGRP interface. The following example shows an outbound policy for GigabitEthernet interface 0/2/0/3:

```
router eigrp 100
    address-family ipv4
    interface GigabitEthernet0/2/0/3
    route-policy if-filter-policy-out out
```

Redistribute

The redistribute attach point in EIGRP allows you to filter redistributed routes from other routing protocols and modify some routing parameters before installing the route in the EIGRP database. The following example shows a policy filter redistribution of RIP routes into EIGRP.

```
router-policy redistribute-rip
    if destination in (100.1.1.0/24) then
        set eigrp-metric 5000000 4000 150 30 2000
    else
        set tag 200
    endif
end-policy
router eigrp 100
    address-family ipv4
    redistribute rip route-policy redistribute-rip
```
### EIGRP Attributes and Operators

This table summarizes the EIGRP attributes and operators per attach points.

**Table 6: EIGRP Attributes and Operators**

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-accept-in</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>default-accept-out</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>if-policy-in</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>eigrp-metric</td>
<td>n/a</td>
<td>add, set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>set</td>
</tr>
<tr>
<td>if-policy-out</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>protocol</td>
<td>is, in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>eigrp-metric</td>
<td>n/a</td>
<td>add, set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>set</td>
</tr>
<tr>
<td>policy-in</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>eigrp-metric</td>
<td>n/a</td>
<td>add, set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>set</td>
</tr>
<tr>
<td>policy-out</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>protocol</td>
<td>is, in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>eigrp-metric</td>
<td>n/a</td>
<td>add, set</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>set</td>
</tr>
</tbody>
</table>
RIP Policy Attach Points

This section describes each of the RIP policy attach points and provides a summary of the RIP attributes and operators.

Default-Information Originate

The default-information originate attach point allows you to conditionally inject the default route 0.0.0.0/0 into RIP updates by evaluating the attached policy. If any routes in the local RIB pass the policy, then the default route is inserted.

The following example shows how to generate a default route if any of the routes that match 10.0.0.0/8 ge 8 le 25 are present in the RIB:

```
route-policy rip-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 25) then
    pass
  endif
end-policy
router rip
  default-information originate route-policy rip-originate
```

Redistribute

The redistribution attach point within RIP allows you to inject routes from other routing protocol sources into the RIP database.

The following example shows how to inject OSPF routes into RIP:

```
route-policy redist-ospf
  set rip-metric 5
end-policy
router rip
  redistribute ospf 1 route-policy redist-ospf
```
Global-Inbound

The global-inbound attach point for RIP allows you to filter or update inbound RIP routes that match a route policy.

The following example shows how to filter the inbound RIP routes that match the route policy named rip-in:

```
router rip
  route-policy rip-in in
```

Global-Outbound

The global-outbound attach point for RIP allows you to filter or update outbound RIP routes that match a route policy.

The following example shows how to filter the outbound RIP routes that match the route policy named rip-out:

```
router rip
  route-policy rip-out out
```

Interface-Inbound

The interface-inbound attach point allows you to filter or update inbound RIP routes that match a route policy for a specific interface.

The following example shows how to filter inbound RIP routes that match the route policy for interface 0/1/0/1:

```
router rip
  interface GigabitEthernet0/1/0/1
    route-policy rip-in in
```

Interface-Outbound

The interface-outbound attach point allows you to filter or update outbound RIP routes that match a route policy for a specific interface.

The following example shows how to filter outbound RIP routes that match the route policy for interface 0/2/0/1:

```
router rip
  interface GigabitEthernet0/2/0/1
    route-policy rip-out out
```

RIP Attributes and Operators

This table summarizes the RIP attributes and operators per attach points.
Table 7: RIP Attributes and Operators

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-information originate</td>
<td>next-hop</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rip-tag</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rib-has-route</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td>global-inbound</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>add</td>
</tr>
<tr>
<td>global-outbound</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>protocol</td>
<td>is, in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>add</td>
</tr>
<tr>
<td>interface-inbound</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>add</td>
</tr>
<tr>
<td>interface-outbound</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>protocol</td>
<td>is, in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>add</td>
</tr>
<tr>
<td>redistribute</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
</tbody>
</table>
PIM Policy Attach Points

This section describes the PIM policy rpf-topology attach point and provides a summary of the PIM attributes and operators.

**rpf-topology**

The rpf-topology attach point is to set the Reverse Path Forwarding (RPF) to any default or non-default tables for particular Sources and/or Groups.

For example, the following policy sets the rpf-topology to table t201 if the destination address is either 225.0.0.1 or 225.0.0.11 then table t201 will be used to figure out the reverse path, else if the destination is 225.0.0.3 or 225.0.0.13 default table will be used to figure out the reverse forwarding path.

```plaintext
route-policy mt4-p1
  if destination in (225.0.0.1, 225.0.0.11) then
    set rpf-topology ipv4 multicast topology t201
  elseif destination in (225.0.0.3, 225.0.0.13) then
    pass
  endif
end-policy

router pim
  rpf topology route-policy mt4-p1
```

To set an extranet VRF for RPF lookups, use the rpf topology route-policy route-policy name command in PIM. For example:

```plaintext
router pim
  rpf topology route-policy extranet-vrf-policy

route-policy extranet-vrf-policy
  if destination in (225.1.1.1/32) then
    set rpf-topology vrf parent-vrf
  else
```

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tag</td>
<td>is, eq, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>route-type</td>
<td>is</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>next-hop</td>
<td>in</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>route-has-level</td>
<td>route-has-level</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rib-metric</td>
<td>is, eq, ge, le</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rip-metric</td>
<td>n/a</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>rip-tag</td>
<td>n/a</td>
<td>set</td>
</tr>
</tbody>
</table>
Note
Set the rpf-topology attribute for PIM to default VRF to extranet the IGMP joins for Multicast VRF Override to function.

PIM Attributes and Operators

This table summarizes the PIM attributes and operators for the rpf-topology attach point.

Table 8: PIM Attributes and Operators

<table>
<thead>
<tr>
<th>Attach Point</th>
<th>Attribute</th>
<th>Match</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpf-topology</td>
<td>destination</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>in</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>rpf-topology</td>
<td>n/a</td>
<td>set</td>
</tr>
</tbody>
</table>

Attached Policy Modification

Policies that are in use do, on occasion, need to be modified. In the traditional configuration model, a policy modification would be done by completely removing the policy and re-entering it. However, this model allows for a window of time in which no policy is attached and default actions to be used, which is an opportunity for inconsistencies to exist. To close this window of opportunity, you can modify a policy in use at an attach point by respecifying it, which allows for policies that are in use to be changed, without having a window of time in which no policy is applied at the given attach point.

Note
A route policy or set that is in use at an attach point cannot be removed because this removal would result in an undefined reference. An attempt to remove a route policy or set that is in use at an attach point results in an error message to the user.

Nonattached Policy Modification

As long as a given policy is not attached at an attach point, the policy is allowed to refer to nonexistent sets and policies. Configurations can be built that reference sets or policy blocks that are not yet defined, and then later those undefined policies and sets can be filled in. This method of building configurations gives much greater flexibility in policy definition. Every piece of policy you want to reference while defining a policy need not exist in the configuration. Thus, you can define a policy sample1 that references a policy sample2 using an apply statement even if the policy sample2 does not exist. Similarly, you can enter a policy statement that refers to a nonexistent set.
However, the existence of all referenced policies and sets is enforced when a policy is attached. Thus, if a user attempts to attach the policy sample1 with the reference to an undefined policy sample2 at an inbound BGP policy using the statement `neighbor 1.2.3.4 address-family ipv4 unicast policy sample1 in`, the configuration attempt is rejected because the policy sample2 does not exist.

Editing Routing Policy Configuration Elements

RPL is based on statements rather than on lines. That is, within the begin-end pair that brackets policy statements from the CLI, a new line is merely a separator, the same as a space character.

The CLI provides the means to enter and delete route policy statements. RPL provides a means to edit the contents of the policy between the begin-end brackets, using a text editor. The following text editors are available on Cisco IOS XR software for editing RPL policies:

- Nano (default)
- Emacs
- Vim

Editing Routing Policy Configuration Elements Using the Nano Editor

To edit the contents of a routing policy using the Nano editor, use the following CLI command in EXEC mode:

```
edit route-policy
    name
    nano
```

A copy of the route policy is copied to a temporary file and the editor is launched. After editing, enter Ctrl-X to save the file and exit the editor. The available editor commands are displayed on screen.

Detailed information on using the Nano editor is available at this URL: [http://www.nano-editor.org/](http://www.nano-editor.org/).

Not all Nano editor features are supported on Cisco IOS XR software.

Editing Routing Policy Configuration Elements Using the Emacs Editor

To edit the contents of a routing policy using the Emacs editor, use the following CLI command in EXEC mode:

```
edit
    route-policy
    name
    emacs
```

Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x
A copy of the route policy is copied to a temporary file and the editor is launched. After editing, save the
editor buffer by using the Ctrl-X and Ctrl-S keystrokes. To save and exit the editor, use the Ctrl-X and Ctrl-C
keystrokes. When you quit the editor, the buffer is committed. If there are no parse errors, the configuration
is committed:

```
RP/0/RP0/CPU0:router# edit route-policy policy_A
----------------------------------------
== MicroEMACS 3.8b () == rpl_edit.139281 ==
  if destination in (2001::/8) then
    drop
  endif
end-policy
!
== MicroEMACS 3.8b () == rpl_edit.139281 ==
Parsing.
83 bytes parsed in 1 sec (82)bytes/sec
Committing.
1 items committed in 1 sec (0)items/sec
Updating.
Updated Commit database in 1 sec
```

If there are parse errors, you are asked whether editing should continue:

```
RP/0/RP0/CPU0:router#edit route-policy policy_B
-- MicroEMACS 3.8b () -- rpl_edit.141738
route-policy policy_B
  set metric-type type_1
  if destination in (2001::/8) then
    drop
  endif
end-policy
!
-- MicroEMACS 3.8b () -- rpl_edit.141738 ==
Parsing.
105 bytes parsed in 1 sec (103)bytes/sec
% Syntax/Authorization errors in one or more commands.!!
% CONFIGURATION FAILED DUE TO SYNTAX/AUTHORIZATION ERRORS
% set metric-type type_1
  if destination in (2001::/8) then
    drop
  endif
end-policy
!
Continue editing? [no]:
```

If you answer yes, the editor continues on the text buffer from where you left off. If you answer no, the running
configuration is not changed and the editing session is ended.

**Editing Routing Policy Configuration Elements Using the Vim Editor**

Editing elements of a routing policy with Vim (Vi IMproved) is similar to editing them with Emacs except
for some feature differences such as the keystrokes to save and quit. To write to a current file and exit, use
the :wq or :x or ZZ keystrokes. To quit and confirm, use the :q keystrokes. To quit and discard changes, use
the :q! keystrokes.

You can reference detailed online documentation for Vim at this URL: [http://www.vim.org/](http://www.vim.org/)
**Editing Routing Policy Configuration Elements Using CLI**

The CLI allows you to enter and delete route policy statements. You can complete a policy configuration block by entering applicable commands such as `end-policy` or `end-set`. Alternatively, the CLI interpreter allows you to use the `exit` command to complete a policy configuration block. The `abort` command is used to discard the current policy configuration and return to global configuration mode.

**Editing Routing Policy Language set elements Using XML**

RPL supports editing set elements using XML. Entries can be appended, prepended, or deleted to an existing set without replacing it through XML.

**Hierarchical Policy Conditions**

The Hierarchical Policy Conditions feature enables the ability to specify a route policy within the "if" statement of another route policy. This ability enables route-policies to be applied for configurations that are based on hierarchical policies.

With the Hierarchical Policy Conditions feature, Cisco IOS XR RPL supports Apply Condition policies that can be used with various types of Boolean operators along with various other matching statements.

**Apply Condition Policies**

Apply Condition policies, which Cisco IOS XR RPL supports, allow usage of a route-policy within an "if" statement of another route-policy.

Consider route-policy configurations `Parent`, `Child A`, and `Child B`:

```plaintext
route-policy Child A
  if destination in (10.10.0.0/16) then
    set local-pref 111
  endif
end-policy

route-policy Child B
  if as-path originates-from '222' then
    set community (333:222) additive
  endif
end-policy

route-policy Parent
  if apply Child A and apply Child B then
    set community (333:333) additive
  else
    set community (333:444) additive
  endif
end-policy
```

In the above scenarios, whenever the policy `Parent` is executed, the decision of the "if" condition in that is selected based on the result of policies `Child A` and `Child B`. The policy `Parent` is equivalent to policy `merged` as given below:

```plaintext
route-policy merged
  if destination in (10.10.0.0/16) and as-path originates-from '222' then
    set local-pref 111
    set community (333:222, 333:333) additive
  elseif destination in (10.10.0.0/16) then /*Only Policy Child A is pass */
    set local-pref 111
```

Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2.x
set community (333:444) additive /*From else block */
elseif as-path originates-from '222' then /*Only Policy Child B is pass */
    set community (333:222, 333:444) additive /*From else block */
else
    set community (333:444) additive /*From else block */
endif
end-policy

Apply Conditions can be used with parameters and are supported on all attach points and on all clients. Hierarchical Apply Conditions can be used without any constraints on a cascaded level.

Existing route policy semantics can be expanded to include this Apply Condition:

Route-policy policy_name
If apply policyA and apply policyB then
    Set med 100
Else if not apply policyD then
    Set med 200
Else
    Set med 300
Endif
End-policy

**Behavior of pass/drop/done RPL Statements for Simple Hierarchical Policies**

This table describes the behavior of **pass**/**drop**/**done** RPL statements, with a possible sequence for executing the **done** statement for Simple Hierarchical Policies.

<table>
<thead>
<tr>
<th>Route-policies with simple hierarchical policies</th>
<th>Possible done statement execution sequence</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass</td>
<td>pass</td>
<td>Marks the prefix as &quot;acceptable&quot; and continues with execution of continue_list statements.</td>
</tr>
<tr>
<td></td>
<td>Continue_list</td>
<td></td>
</tr>
<tr>
<td>drop</td>
<td>Stmts_list</td>
<td>Rejects the route immediately on hitting the <strong>drop</strong> statement and stops policy execution.</td>
</tr>
<tr>
<td></td>
<td><strong>drop</strong></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>Stmts_list</td>
<td>Accepts the route immediately on hitting the <strong>done</strong> statement and stops policy execution.</td>
</tr>
<tr>
<td></td>
<td><strong>done</strong></td>
<td></td>
</tr>
<tr>
<td>pass followed by done</td>
<td>pass</td>
<td>Exits immediately at the <strong>done</strong> statement with &quot;accept route&quot;.</td>
</tr>
<tr>
<td></td>
<td>Statement_list</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>done</strong></td>
<td></td>
</tr>
<tr>
<td>drop followed by done</td>
<td>drop</td>
<td>This is an invalid scenario at execution point of time. Policy terminates execution at the <strong>drop</strong> statement itself, without going through the statement list or the <strong>done</strong> statement; the prefix will be rejected or dropped.</td>
</tr>
<tr>
<td></td>
<td>Statement_list</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>done</strong></td>
<td></td>
</tr>
</tbody>
</table>
Behavior of pass/drop/done RPL Statements for Hierarchical Policy Conditions

This section describes the behavior of pass/drop/done RPL statements, with a possible sequence for executing the done statement for Hierarchical Policy Conditions.

Terminology for policy execution: "true-path", "false-path", and "continue-path".

Route-policy parent
If apply hierarchical_policy_condition then
  TRUE-PATH : if hierarchical_policy_condition returns TRUE then this path will be executed.
  Else
  FALSE-PATH : if hierarchical_policy_condition returns FALSE then this path will be executed.
  End-if
CONTINUE-PATH : Irrespective of the TRUE/FALSE this path will be executed.
End-policy

<table>
<thead>
<tr>
<th>Hierarchical policy conditions</th>
<th>Possible done statement execution sequence</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass</td>
<td>pass Continue_list</td>
<td>Marks the return value as &quot;true&quot; and continues execution within the same policy condition. If there is no statement after &quot;pass&quot;, returns &quot;true&quot;.</td>
</tr>
<tr>
<td>pass followed by done</td>
<td>pass or set Stmt_list done</td>
<td>Marks the return value as &quot;true&quot; and continues execution till the done statement. Returns &quot;true&quot; to the apply policy condition to take &quot;true-path&quot;.</td>
</tr>
<tr>
<td>done</td>
<td>Stmt_list without pass set operation DONE</td>
<td>Returns &quot;false&quot;. Condition takes &quot;false-path&quot;.</td>
</tr>
<tr>
<td>drop</td>
<td>Stmt_list drop Stmt_list</td>
<td>The prefix is dropped or rejected.</td>
</tr>
</tbody>
</table>

Nested Wildcard Apply Policy

The hierarchical constructs of Routing Policy Language (RPL) allows one policy to refer to another policy. The referred or called policy is known as a child policy. The policy from which another policy is referred is called calling or parent policy. A calling or parent policy can nest multiple child policies for attachment to a common set of BGP neighbors. The nested wildcard apply policy allows wildcard (*) based apply nesting. The wildcard operation permits declaration of a generic apply statement that calls all policies that contain a specific defined set of alphanumeric characters, defined on the router.
A wildcard is specified by placing an asterisk (*) at the end of the policy name in an apply statement. Passing parameters to wildcard policy is not supported. The wildcard indicates that any value for that portion of the apply policy matches.

To illustrate nested wildcard apply policy, consider this policy hierarchy:

```
routing-policy Nested_Wildcard
  apply service_policy_customer*
  end-policy

  routing-policy service_policy_customer_a
  if destination in prfx_set_Customer_A then
    set extcommunity rt (1:1) additive
  endif
  end-policy

  routing-policy service_policy_customer_b
  if destination in prfx_set_Customer_B then
    set extcommunity rt (1:1) additive
  endif
  end-policy

  routing-policy service_policy_customer_c
  if destination in prfx_set_Customer_C then
    set extcommunity rt (1:1) additive
  endif
  end-policy
```

Here, a single parent apply statement (apply service_policy_customer*) calls (inherits) all child polices that contain the identified character string "service_policy_customer". As each child policy is defined globally, the parent dynamically nests the child policies based on the policy name. The parent is configured once and inherits each child policy on demand. There is no direct association between the parent and the child policies beyond the wildcard match statement.

### Wildcards for Route Policy Sets

Route policies are defined in a modular form, and comprise of sets of comparative statements. Using wildcards to define a range of sets, significantly reduces the complexity of a policy.

Wildcards can be used to define a range of prefix sets, community sets, AS-path sets, or extended community sets. For information on using wildcards in policy sets, see Using Wildcards For Routing Policy Sets: Example, on page 81.

### How to Implement Routing Policy

This section contains the following procedures:

#### Defining a Route Policy

This task explains how to define a route policy.
Note

- If you want to modify an existing routing policy using the command-line interface (CLI), you must redefine the policy by completing this task.
- Modifying the RPL scale configuration may take a long time.
- BGP may crash either due to large scale RPL configuration changes, or during consecutive RPL changes. To avoid BGP crash, wait until there are no messages in the BGP In/Out queue before committing further changes.

### SUMMARY STEPS

1. `configure`
2. `route-policy name [parameter1, parameter2, ..., parameterN]`
3. `end-policy`
4. `commit`

### DETAILED STEPS

<table>
<thead>
<tr>
<th>Command or Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong> configure</td>
<td>Enters route-policy configuration mode.</td>
</tr>
<tr>
<td><strong>Step 2</strong> route-policy name [parameter1, parameter2, ..., parameterN]</td>
<td>Enters route-policy configuration mode.</td>
</tr>
<tr>
<td><strong>Example:</strong> RP/0/RP0/CPU0:router(config)# route-policy sample1</td>
<td></td>
</tr>
<tr>
<td><strong>Step 3</strong> end-policy</td>
<td>Ends the definition of a route policy and exits route-policy configuration mode.</td>
</tr>
<tr>
<td><strong>Example:</strong> RP/0/RP0/CPU0:router(config-rpl)# end-policy</td>
<td></td>
</tr>
<tr>
<td><strong>Step 4</strong> commit</td>
<td></td>
</tr>
</tbody>
</table>

## Attaching a Routing Policy to a BGP Neighbor

This task explains how to attach a routing policy to a BGP neighbor.

### Before You Begin

A routing policy must be preconfigured and well defined prior to it being applied at an attach point. If a policy is not predefined, an error message is generated stating that the policy is not defined.
SUMMARY STEPS

1. configure
2. router bgp as-number
3. neighbor ip-address
4. address-family { ipv4 unicast | ipv4 multicast | ipv4 labeled-unicast | ipv4 tunnel | ipv4 mdt | ipv6 unicast | ipv6 multicast | ipv6 labeled-unicast | vpng4 unicast | vpng6 unicast }
5. route-policy policy-name { in | out }
6. commit

DETAILED STEPS

<table>
<thead>
<tr>
<th>Command or Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong> configure</td>
<td>Configures a BGP routing process and enters router configuration mode.</td>
</tr>
<tr>
<td><strong>Step 2</strong> router bgp as-number</td>
<td>Configures a BGP routing process and enters router configuration mode.</td>
</tr>
<tr>
<td>Example:</td>
<td>• The <em>as-number</em> argument identifies the autonomous system in which the router resides. Valid values are from 0 to 65535. Private autonomous system numbers that can be used in internal networks range from 64512 to 65535.</td>
</tr>
<tr>
<td>neighbor ip-address</td>
<td>Specifies a neighbor IP address.</td>
</tr>
<tr>
<td>Example:</td>
<td></td>
</tr>
</tbody>
</table>
| address-family { ipv4 unicast | ipv4 multicast | ipv4 labeled-unicast | ipv4 tunnel | ipv4 mdt | ipv6 unicast | ipv6 multicast | ipv6 labeled-unicast | vpng4 unicast | vpng6 unicast }
| Example:                          | Specifies the address family.                                           |
| route-policy policy-name { in | out }                               | Attaches the route-policy, which must be well formed and predefined.    |
| Example:                          |                                                                         |
| commit                            |                                                                         |
Modifying a Routing Policy Using a Text Editor

This task explains how to modify an existing routing policy using a text editor. See Editing Routing Policy Configuration Elements, on page 70 for information on text editors.

SUMMARY STEPS

1. edit { route-policy | prefix-set | as-path-set | community-set | extcommunity-set { rt | soo } | policy-global | rd-set } name [ nano | emacs | vim | inline { add | prepend | remove } set-element }

2. show rpl route-policy [ name [ detail ] | states | brief ]

3. show rpl prefix-set [ name | states | brief ]

DETAILED STEPS

<table>
<thead>
<tr>
<th>Command or Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong> edit { route-policy</td>
<td>prefix-set</td>
</tr>
<tr>
<td></td>
<td>• A copy of the route policy, prefix set, AS path set, community set, or extended community set is copied to a temporary file and the editor is launched.</td>
</tr>
<tr>
<td></td>
<td>• After editing with Nano, save the editor buffer and exit the editor by using the Ctrl-X keystroke.</td>
</tr>
<tr>
<td></td>
<td>• After editing with Emacs, save the editor buffer by using the Ctrl-X and Ctrl-S keystrokes. To save and exit the editor, use the Ctrl-X and Ctrl-C keystrokes.</td>
</tr>
<tr>
<td></td>
<td>• After editing with Vim, to write to a current file and exit, use the :wq or :x or ZZ keystrokes. To quit and confirm, use the :q keystrokes. To quit and discard changes, use the :q! keystrokes.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td>RP/0/RP0/CPU0:router# edit route-policy sample1</td>
<td></td>
</tr>
<tr>
<td><strong>Step 2</strong> show rpl route-policy [ name [ detail ]</td>
<td>states</td>
</tr>
<tr>
<td></td>
<td>• Use the detail keyword to display all policies and sets that a policy uses.</td>
</tr>
<tr>
<td></td>
<td>• Use the states keyword to display all unused, inactive, and active states.</td>
</tr>
<tr>
<td></td>
<td>• Use the brief keyword to list the names of all extended community sets without their configurations.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td>RP/0/RP0/CPU0:router# show rpl route-policy sample2</td>
<td></td>
</tr>
<tr>
<td><strong>Step 3</strong> show rpl prefix-set [ name</td>
<td>states</td>
</tr>
<tr>
<td></td>
<td>• To display the contents of a named AS path set, community set, or extended community set, replace the prefix-set keyword with as-path-set, community-set, or extcommunity-set, respectively.</td>
</tr>
</tbody>
</table>
### Configuration Examples for Implementing Routing Policy

This section provides the following configuration examples:

#### Routing Policy Definition: Example

In the following example, a BGP route policy named sample1 is defined using the `route-policy name` command. The policy compares the network layer reachability information (NLRI) to the elements in the prefix set test. If it evaluates to true, the policy performs the operations in the `then` clause. If it evaluates to false, the policy performs the operations in the `else` clause, that is, sets the MED value to 200 and adds the community 2:100 to the route. The final steps of the example commit the configuration to the router, exit configuration mode, and display the contents of route policy sample1.

```bash
configure
route-policy sample1
  if destination in test then
    drop
  else
    set med 200
    set community (2:100) additive
  endif
end-policy
end
show config running route-policy sample1
Building configuration...
route-policy sample1
  if destination in test then
    drop
  else
    set med 200
    set community (2:100) additive
  endif
end-policy
```

#### Simple Inbound Policy: Example

The following policy discards any route whose network layer reachability information (NLRI) specifies a prefix longer than /24, and any route whose NLRI specifies a destination in the address space reserved by RFC 1918. For all remaining routes, it sets the MED and local preference, and adds a community to the list in the route.

For routes whose community lists include any values in the range from 101:202 to 106:202 that have a 16-bit tag portion containing the value 202, the policy prepends autonomous system number 2 twice, and adds the

---

**Example:**

RP/0/RP0/CPU0:router# show rpl
prefix-set prefixset1
community 2:666 to the list in the route. Of these routes, if the MED is either 666 or 225, then the policy sets the origin of the route to incomplete, and otherwise sets the origin to IGP.

For routes whose community lists do not include any of the values in the range from 101:202 to 106:202, the policy adds the community 2:999 to the list in the route.

```
prefix-set too-specific
  0.0.0.0/0 ge 25 le 32
end-set

prefix-set rfc1918
  10.0.0.0/8 le 32,
  172.16.0.0/12 le 32,
  192.168.0.0/16 le 32
end-set

route-policy inbound-tx
  if destination in too-specific or destination in rfc1918 then
    drop
  endif
  set med 1000
  set local-preference 90
  set community (2:1001) additive
  if community matches-any ([101..106]:202) then
    prepend as-path 2.30 2
  endif
  set community (2:666) additive
  if med is 666 or med is 225 then
    set origin incomplete
  else
    set origin igp
  endif
  else
    set community (2:999) additive
  endif
end-policy
```

```
routing bgp 2
  neighbor 10.0.1.2 address-family ipv4 unicast route-policy inbound-tx in
```

# Modular Inbound Policy: Example

The following policy example shows how to build two inbound policies, in-100 and in-101, for two different peers. In building the specific policies for those peers, the policy reuses some common blocks of policy that may be common to multiple peers. It builds a few basic building blocks, the policies common-inbound, filter-bogons, and set-lpref-prepend.

The filter-bogons building block is a simple policy that filters all undesirable routes, such as those from the RFC 1918 address space. The policy set-lpref-prepend is a utility policy that can set the local preference and prepend the AS path according to parameterized values that are passed in. The common-inbound policy uses these filter-bogons building blocks to build a common block of inbound policy. The common-inbound policy is used as a building block in the construction of in-100 and in-101 along with the set-lpref-prepend building block.

This is a simple example that illustrates the modular capabilities of the policy language.

```
prefix-set bogon
  10.0.0.0/8 ge 8 le 32,
  0.0.0.0/0 ge 27 le 32,
  192.168.0.0/16 ge 16 le 32
end-set

route-policy in-100
  apply common-inbound
```

```
prefix-set in-100
  0.0.0.0 ge 8 le 32
end-set
```

```
route-policy in-101
  apply common-inbound
```

```
!```
if community matches-any ([100..120]:135) then
    apply set-lpref-prepend (100,100,2)
    set community (2:1234) additive
else
    set local-preference 110
endif
if community matches-any ([100..666]:[100..999]) then
    set med 444
    set local-preference 200
    set community (no-export) additive
endif
end-policy
!
route-policy in-101
apply common-inbound
if community matches-any ([101..200]:201) then
    apply set-lpref-prepend(100,101,2)
    set community (2:1234) additive
else
    set local-preference 125
endif
end-policy
!
route-policy filter-bogons
if destination in bogon then
    drop
else
    pass
endif
end-policy
!
route-policy common-inbound
    apply filter-bogons
    set origin igp
    set community (2:333)
end-policy
!
route-policy set-lpref-prepend($lpref,$as,$prependcnt)
    set local-preference $lpref
    prepend as-path $as $prependcnt
end-policy

Using Wildcards For Routing Policy Sets: Example

This section describes examples of configuring routing policy sets with wildcards.

Using Wildcards for Prefix Sets

Use the following example to configure a routing policy with wildcards for prefix sets.

1 Configure the required prefix sets in the global configuration mode.

    RP/0/0/CPU0:ios(config)# prefix-set pfx_set1
    RP/0/0/CPU0:ios(config-pfx)# 1.2.3.4/32
    RP/0/0/CPU0:ios(config-pfx)# end-set
    RP/0/0/CPU0:ios(config-pfx)# prefix-set pfx_set2
    RP/0/0/CPU0:ios(config-pfx)# 2.2.2.2/32
    RP/0/0/CPU0:ios(config-pfx)# end-set

2 Configure a route policy with wildcards to refer to the prefix sets.

    RP/0/0/CPU0:ios(config)# route-policy WILDCARD_PREFIX_SET
    RP/0/0/CPU0:ios(config-rpl)# if destination in prefix-set* then pass else drop endif
    RP/0/0/CPU0:ios(config-rpl)# end-policy

This route policy configuration accepts routes with the prefixes mentioned in the two prefix sets, and drops all other non-matching routes.
Implementing Routing Policy

Using Wildcards For Routing Policy Sets: Example

3 Commit your configuration.

RP/0/0/CPU0:ios(config)# commit

This completes the configuration of routing policy with wildcards for prefix sets. For detailed information on prefix sets, see prefix-set, on page 10.

Using Wildcards for AS-Path Sets

Use the following example to configure a routing policy with wildcards for AS-path sets.

1 Configure the required AS-path sets in the global configuration mode.

```
RP/0/0/CPU0:ios(config)# as-path-set AS_SET1
RP/0/0/CPU0:ios(config-as)# ios-regex '_22$'
RP/0/0/CPU0:ios(config-as)# ios-regex '_25$'
RP/0/0/CPU0:ios(config-as)# end-set
RP/0/0/CPU0:ios(config)# as-path-set AS_SET2
RP/0/0/CPU0:ios(config-as)# ios-regex '_42$'
RP/0/0/CPU0:ios(config-as)# ios-regex '_47$'
RP/0/0/CPU0:ios(config-as)# end-set
```

2 Configure a route policy with wildcards to refer to the AS-path sets.

```
RP/0/0/CPU0:ios(config)# route-policy WILDCARD_AS_SET
RP/0/0/CPU0:ios(config-rpl)# if as-path in as-path-set* then pass else drop endif
RP/0/0/CPU0:ios(config-rpl)# end-policy
```

This route policy configuration accepts routes with AS-path attributes as mentioned in the two AS-path sets, and drops all other non-matching routes.

3 Commit your configuration.

RP/0/0/CPU0:ios(config)# commit

This completes the configuration of routing policy with wildcards for AS-path sets. For detailed information on AS-path sets, see as-path-set, on page 5.

Using Wildcards for Community Sets

Use the following example to configure a routing policy with wildcards for community sets.

1 Configure the required community sets in the global configuration mode.

```
RP/0/0/CPU0:ios(config)# community-set CSET1
RP/0/0/CPU0:ios(config-comm)# 12:24,
RP/0/0/CPU0:ios(config-comm)# 12:36,
RP/0/0/CPU0:ios(config-comm)# 12:72
RP/0/0/CPU0:ios(config-comm)# end-set
RP/0/0/CPU0:ios(config)# community-set CSET2
RP/0/0/CPU0:ios(config-comm)# 24:12,
RP/0/0/CPU0:ios(config-comm)# 24:42,
RP/0/0/CPU0:ios(config-comm)# 24:64
RP/0/0/CPU0:ios(config-comm)# end-set
```

2 Configure a route policy with wildcards to refer to the community sets.

```
RP/0/0/CPU0:ios(config)# route-policy WILDCARD_COMMUNITY_SET
RP/0/0/CPU0:ios(config-rpl)# if community matches-any community-set* then pass else drop endif
RP/0/0/CPU0:ios(config-rpl)# end-policy
```

This route policy configuration accepts routes with community set values as mentioned in the two community sets, and drops all other non-matching routes.

3 Commit your configuration.

RP/0/0/CPU0:ios(config)# commit

This completes the configuration of routing policy with wildcards for community sets. For detailed information on community path sets, see community-set, on page 6.
Using Wildcards for Extended Community Sets

Use the following example to configure a routing policy with wildcards for extended community sets.

1 Configure the required extended community sets in the global configuration mode.

```
RP/0/0/CPU0:ios(config)# extcommunity-set rt RT_SET1
RP/0/0/CPU0:ios(config-ext)# 1.2.3.4:555,
RP/0/0/CPU0:ios(config-ext)# 1234:555
RP/0/0/CPU0:ios(config-ext)# end-set
RP/0/0/CPU0:ios(config)# extcommunity-set rt RT_SET2
RP/0/0/CPU0:ios(config-ext)# 1.1.1.1:777,
RP/0/0/CPU0:ios(config-ext)# 1111:777
RP/0/0/CPU0:ios(config-ext)# end-set
```

2 Configure a route policy with wildcards to refer to the extended community sets.

```
RP/0/0/CPU0:ios(config)# route-policy WILDCARD_EXT_COMMUNITY_SET
RP/0/0/CPU0:ios(config-rpl)# if extcommunity rt matches-any extcommunity-set* then pass
else drop endif
RP/0/0/CPU0:ios(config-rpl)# end-policy
```

This route policy configuration accepts routes with extended community set values as mentioned in the two extended community sets, and drops all other non-matching routes.

3 Commit your configuration.

```
RP/0/0/CPU0:ios(config)# commit
```

This completes the configuration of routing policy with wildcards for extended community sets. For detailed information on extended community path sets, see extcommunity-set, on page 7.

Translating Cisco IOS Route Maps to Cisco IOS XR Routing Policy Language: Example

RPL performs the same functions as route-maps. See Converting Cisco IOS Configurations to Cisco IOS XR Configurations.

Additional References

The following sections provide references related to implementing RPL.

### Related Documents

<table>
<thead>
<tr>
<th>Related Topic</th>
<th>Document Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing policy language commands: complete command syntax, command modes, command history, defaults, usage guidelines, and examples</td>
<td>Routing Policy Language Commands on Cisco IOS XR Software module of the Cisco IOS XR Routing Command Reference for the Cisco CRS Router</td>
</tr>
<tr>
<td>Regular expression syntax</td>
<td>Understanding Regular Expressions, Special Characters and Patterns appendix in the Cisco IOS XR Getting Started Guide for the Cisco CRS Router</td>
</tr>
</tbody>
</table>
### Standards

<table>
<thead>
<tr>
<th>Standards</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>No new or modified standards are supported by this feature, and support for existing standards has not been modified by this feature.</td>
<td>—</td>
</tr>
</tbody>
</table>

### MIBs

<table>
<thead>
<tr>
<th>MIBs</th>
<th>MIBs Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>To locate and download MIBs using Cisco IOS XR software, use the Cisco MIB Locator found at the following URL and choose a platform under the Cisco Access Products menu: <a href="http://cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml">http://cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml</a></td>
</tr>
</tbody>
</table>

### RFCs

<table>
<thead>
<tr>
<th>RFCs</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 1771</td>
<td>A Border Gateway Protocol 4 (BGP-4)</td>
</tr>
<tr>
<td>RFC 4360</td>
<td>BGP Extended Communities Attribute</td>
</tr>
</tbody>
</table>

### Technical Assistance

<table>
<thead>
<tr>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Cisco Technical Support website contains thousands of pages of searchable technical content, including links to products, technologies, solutions, technical tips, and tools. Registered Cisco.com users can log in from this page to access even more content.</td>
<td><a href="http://www.cisco.com/techsupport">http://www.cisco.com/techsupport</a></td>
</tr>
</tbody>
</table>