

Chapter 5

Configure Extended Match Conditions

This chapter describes how to configure extended match conditions for a routing policy. In general, the extended match conditions include criteria that are defined separately from the routing policy (autonomous system [AS] path regular expressions, communities, and prefix lists) and are more complex than standard match conditions. You should have an in-depth understanding of extended match conditions before configuring them. For information about standard match conditions, see Table 9 on page 44.

This chapter describes how to configure the following extended match conditions:

Configure AS Path Regular Expressions on page 89

Configure Communities on page 96

Configure Prefix Lists on page 108

Configure Route Lists on page 112

Configure Subroutines on page 120

Configure AS Path Regular Expressions

A Border Gateway Protocol (BGP) *AS path* is a path to a destination. An AS path consists of AS numbers of networks that a packet traverses if it takes the associated route to a destination. The AS numbers are assembled in a sequence from left to right, for example, AS5, AS4, AS3, AS2, AS1. For a packet to reach a destination using this route, it first traverses AS5 and so on until it reaches AS1, which is the last AS before its destination.

You can define a match condition based on all or portions of the AS path. To do this, you create a named AS path regular expression and then include it in a routing policy.

This section describes the following tasks for configuring AS path regular expressions and provides the following examples:

Define AS Path Regular Expressions on page 90

How AS Path Regular Expressions Are Evaluated on page 94

Examples: Configure AS Path Regular Expressions on page 95

Define AS Path Regular Expressions

You can create a named AS path regular expression and then include it in a routing policy with the as-path match condition (described in Table 9 on page 44). To create a named AS path regular expression, include the as-path statement:

```
as-path name regular-expression;
```

You can include this statement at the following hierarchy levels:

```
[edit policy-options]
```

```
[edit logical-routers logical-router-name policy-options]
```

To include the AS path regular expression in a routing policy, include the as-path match condition in the from statement:

```
as-path name regular-expression;
policy-statement policy-name {
  term term-name {
    from {
      as-path names;
    }
  }
}
```

Additionally, you can create a named AS path group made up of AS path regular expressions and then include it in a routing policy with the as-path-group match condition. To create a named AS path group, include the as-path-group statement:

```
as-path-group group-name {
  as-path name [ regular-expressions ];
}
```

You can include this statement at the following hierarchy levels:

```
[edit policy-options]
```

```
[edit logical-routers logical-router-name policy-options]
```

To include the AS path regular expressions within the AS path group in a routing policy, include the as-path-group match condition in the from statement:

```
[edit policy-options]
as-path-group group-name {
  as-path name [ regular-expressions ];
}
policy-statement policy-name {
  term term-name {
    from {
      as-path-group group-name;
    }
  }
}
```



NOTE: You cannot have both `as-path` and `as-path-group` in the same policy term.



NOTE: You can include the names of multiple AS path regular expressions in the `as-path` match condition in the `from` statement. If you do this, only one AS path regular expression needs to match for a match to occur. The AS path regular expression matching is effectively a logical OR operation.

The AS path name identifies the regular expression. It can contain letters, numbers, and hyphens (-), and can be up to 255 characters. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

The regular expression is used to match all or portions of the AS path. It consists of two components, which you specify in the following format:

term <*operator*>

term—Identifies an AS. You can specify it in one of the following ways:

AS number—The entire AS number composes one *term*. You cannot reference individual characters within an AS number, which differs from regular expressions as defined in POSIX 1003.2.

Wildcard character—Matches any single AS number. The wildcard character is a period (.). You can specify multiple wildcard characters.

AS path—A single AS number or a group of AS numbers enclosed in parentheses. Grouping the regular expression in this way allows you to perform a common operation on the group as a whole and to give the group precedence. The grouped path can itself include operators.

operator—(Optional) An operator specifying how the term must match. Most operators describe how many times the term must be found to be considered a match (for example, any number of occurrences, or zero or one occurrence). Table 14 lists the regular expression operators supported for AS paths. You place operators immediately after *term* with no intervening space, except for the pipe (|) and dash (-) operators, which you place between two terms, and parentheses, with which you enclose terms.

You can specify one or more *term*-*operator* pairs in a single regular expression.

Table 15 shows examples of how to define regular expressions to match AS paths.

Table 14: AS Path Regular Expression Operators

Operator	Match...
{ <i>m,n</i> }	At least <i>m</i> and at most <i>n</i> repetitions of <i>term</i> . Both <i>m</i> and <i>n</i> must be positive integers, and <i>m</i> must be smaller than <i>n</i> .
{ <i>m</i> }	Exactly <i>m</i> repetitions of <i>term</i> . <i>m</i> must be a positive integer.
{ <i>m</i> ,}	<i>m</i> or more repetitions of <i>term</i> . <i>m</i> must be a positive integer.
*	Zero or more repetitions of <i>term</i> . This is equivalent to {0,}.
+	One or more repetitions of <i>term</i> . This is equivalent to {1,}.
?	Zero or one repetition of <i>term</i> . This is equivalent to {0,1}.
	One of the two terms on either side of the pipe.
–	Between a starting and ending range, inclusive.
^	Character at the beginning of an AS path regular expression. This character is added implicitly; therefore, the use of it is optional.
\$	Character at the end of an AS path regular expression. This character is added implicitly; therefore, the use of it is optional.
()	A group of terms that are enclosed in the parentheses. If enclosed in quotation marks with no intervening space (“()”), indicates a null. Intervening space between the parentheses and the terms is ignored.
[]	Set of characters. One character from the set can match. To specify the start and end of a range, use a hyphen (–).
^	Not operator.

Table 15: Examples of Defining AS Path Regular Expressions

AS Path to Match	Regular Expression	Example Matches
AS path is 1234	1234	1234
Zero or more occurrences of AS number 1234	1234*	1234 1234 1234 1234 1234 1234 Null AS path
Zero or one occurrence of AS number 1234	1234? or 1234{0,1}	1234 Null AS path
One through four occurrences of AS number 1234	1234{1,4}	1234 1234 1234 1234 1234 1234 1234 1234 1234 1234
One through four occurrences of AS number 12 followed by one occurrence of AS number 34	12{1,4} 34	12 34 12 12 34 12 12 12 34 12 12 12 12 34
Range of AS numbers to match a single AS number	123–125 [123–125]*	123 or 124 or 125 Null AS path 123 124 124 125 125 125

AS Path to Match	Regular Expression	Example Matches
Path whose second AS number must be 56 or 78	(. 56) (. 78) or . (56 78)	1234 56 34 78
Path whose second AS number might be 56 or 78	. (56 78)?	1234 56 34
Path whose first AS number is 123 and second AS number is either 56 or 78	123 (56 78)	123 56 123 78
Path of any length, except nonexistent, whose second AS number can be anything, including nonexistent	. * or .{0,}	1234 1234 5678 1234 5 6 7 8
AS path is 1 2 3	1 2 3	1 2 3
One occurrence of the AS numbers 1 and 2, followed by one or more occurrences of the number 3	1 2 3+	1 2 3 1 2 3 3 1 2 3 3 3
One or more occurrences of AS number 1, followed by one or more occurrences of AS number 2, followed by one or more occurrences of AS number 3	1+ 2+ 3+	1 2 3 1 1 2 3 1 1 2 2 3 1 1 2 2 3 3
Path of any length that begins with AS numbers 4, 5, 6	4 5 6 . *	4 5 6 4 5 6 7 8 9
Path of any length that ends with AS numbers 4, 5, 6	. * 4 5 6	4 5 6 1 2 3 4 5 6
AS path 5, 12, or 18	[5 12 18]	5 12 18

Null AS Path

You can use AS path regular expressions to create a null AS path that matches routes (prefixes) that have originated in your AS. These routes have not been advertised to your AS by any external peers. To create a null AS path, use the parentheses operator enclosed in quotation marks with no intervening spaces:

"()"

Example: Null AS Path

AS 1 and AS 3 are connected to AS 2, which you administrate. AS 3 advertises its routes to your AS, but you do not want to advertise AS 3 routes to AS 1 and thereby begin routing traffic from AS 1 to AS 3 through your AS. To prevent this situation from occurring, you can configure an export policy for AS 1 (1.2.2.6) that allows routes for your AS to be advertised to AS 1 but does not allow routes for AS 3 or routes for any other connected AS to be advertised to AS 1:

```

[edit policy-options]
as-path null-as "()";
policy-statement only-my-routes {
  term just-my-as {
    from {
      protocol bgp;
      as-path null-as;
    }
    then accept;
  }
  term nothing-else {
    then reject;
  }
}
protocol {
  bgp {
    neighbor 10.2.2.6 {
      export only-my-routes;
    }
  }
}

```

How AS Path Regular Expressions Are Evaluated

AS path regular expressions implement the extended (modern) regular expressions as defined in POSIX 1003.2. They are identical to the UNIX regular expressions with the following exceptions:

The basic unit of matching in an AS path regular expression is the AS number and not an individual character.

A regular expression matches a route only if the AS path in the route exactly matches *regular-expression*. The equivalent UNIX regular expression is *^regular-expression\$*. For example, the AS path regular expression 1234 is equivalent to the UNIX regular expression *^1234\$*.

You can specify a regular expression using wildcard operators.

Examples: Configure AS Path Regular Expressions

Exactly match routes with the AS path 1234 56 78 9 and accept them:

```
[edit]
policy-options {
  as-path wellington "1234 56 78 9";
  policy-statement from-wellington {
    term term1 {
      from as-path wellington;
    }
    then {
      preference 200;
      accept;
    }
    term term2 {
      then reject;
    }
  }
}
```

Match alternate paths to an AS and accept them after modifying the preference:

```
[edit]
policy-options {
  as-path wellington-alternate "1234{1,6} (56|47)? (78|101|112)* 9+";
  policy-statement from-wellington {
    from as-path wellington-alternate;
  }
  then {
    preference 200;
    accept;
  }
}
```

Match routes with an AS path of 123, 124, or 125 and accept them after modifying the preference:

```
[edit]
policy-options {
  as-path addison "123-125";
  policy-statement from-addison {
    from as-path addison;
  }
  then {
    preference 200;
    accept;
  }
}
```

Configure Communities

A *BGP community* is a group of destinations that share a common property. Community information is included as a path attribute in BGP update messages. This information identifies community members and allows you to perform actions on a group without having to elaborate upon each member. You can define match conditions based on a BGP community, which this section describes. For information about actions that can be performed on the community, see Table 11 on page 49.



NOTE: You can assign community tags to non-BGP routes through configuration (for static, aggregate, or generated routes) or an import routing policy. These tags can then be matched when BGP exports the routes.

This section includes the following information:

Define Communities on page 96

How Communities Are Evaluated on page 107

Define Communities

You can create a named community and include it in a routing policy with the community match condition (described in Table 9 on page 44).

You can configure community and extended communities attributes to be included in BGP update messages. The community attribute is only four octets. The BGP extended communities attribute provides a larger range (eight octets) for grouping or categorizing communities. You can use community and extended communities attributes to trigger routing decisions, such as acceptance, rejection, preference, or redistribution.

The *community-ids* format varies according to the type of attribute that you use. The BGP community attribute format is *as-number:community-value*. The BGP extended communities attribute format is *type:administrator:assigned-number*.

When specifying *community-ids* for the community attribute, you can use UNIX-style regular expressions. Regular expressions are not supported for the extended communities attributes.

To define communities, you can do the following:

Configure the Community Attribute on page 97

Configure the Extended Communities Attribute on page 105

Invert Community Matches on page 107

Configure Link Bandwidth on page 107

Configure the Community Attribute

To create a named community and define the community members, include the community statement:

```
community name {
    invert-match;
    members [ community-ids ];
}
```

You can configure the statement at the following hierarchy levels:

```
[edit policy-options]
[edit logical-routers logical-router-name policy-options]
```

To include the community in a routing policy, include the community condition in the from statement:

```
[edit policy-options]
community name members [ community-ids ];
policy-statement policy-name {
    term term-name {
        from {
            community [ names ];
        }
    }
}
```



NOTE: You can include the names of multiple communities in the community match condition in the from statement. If you do this, only one community needs to match for a match to occur. The community matching is effectively a logical OR operation.

name identifies the community or communities. It can contain letters, numbers, and hyphens (-) and can be up to 255 characters long. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

community-ids defines one or more members of the community. It consists of two components, which you specify in the following format:

```
as-number:community-value
```

as-number—AS number of the community member. It can be a value from 1 through 65,534. You can specify the AS number in one of the following ways:

AS number.

Asterisk (*)—A wildcard character that matches all AS numbers. (In the definition of the community attribute, the asterisk also functions as described in Table 16 on page 99.)

Period (.)—A wildcard character that matches any single digit in an AS number.

Group of AS numbers—A single AS number or a group of AS numbers enclosed in parentheses. Grouping the numbers in this way allows you to perform a common operation on the group as a whole and to give the group precedence. The grouped numbers can themselves include regular expression operators. For more information about the community regular expressions, see “Configure the Community Attribute Using UNIX Regular Expressions” on page 99.

community-value—Identifier of the community member. It can be a number from 0 through 65,535. You can specify the community value in one of the following ways:

Community value number.

Asterisk (*)—A wildcard character that matches all community values. (In the definition of the community attribute, the asterisk also functions as described in Table 16 on page 99.)

Period (.)—A wildcard character that matches any single digit in a community value number.

Group of community value numbers—A single community value number or a group of community value numbers enclosed in parentheses. Grouping the regular expression in this way allows you to perform a common operation on the group as a whole and to give the group precedence. The grouped path can itself include regular expression operators.

You also can specify *community-id* as one of the following well-known community names, which are defined in RFC 1997, *BGP Communities A ttribute*:

no-advertise—Routes in this community name must not be advertised to other BGP peers.

no-export—Routes in this community must not be advertised outside a BGP confederation boundary.

no-export-subconfed—Routes in this community must not be advertised to external BGP peers, including peers in other members’ ASs inside a BGP confederation.

Additionally, you can explicitly exclude BGP community information with a static route by using the *none* option. Include this option when configuring an individual route in the route portion to override a community option specified in the defaults portion.

See also “Configure the Extended Communities Attribute” on page 105.

The following section describes the following topics:

Configure the Community Attribute Using UNIX Regular Expressions on page 99

Do Not Advertise Communities to Neighbors on page 100

Examples: Configure the Community Attribute on page 101

Configure the Community Attribute Using UNIX Regular Expressions

When specifying *community-ids*, you can use UNIX-style regular expressions to specify the AS number and the member identifier. A regular expression consists of two components, which you specify in the following format:

term<*operator*>

term identifies the string to match.

operator specifies how the term must match. Table 16 lists the regular expression operators supported for the community attribute. You place an operator immediately after *term* with no intervening space, except for the pipe (|) and dash (–) operators, which you place between two terms, and parentheses, with which you enclose terms. Table 17 shows examples of how to define *community-ids* using community regular expressions. The operator is optional.

Community regular expressions are identical to the UNIX regular expressions. Both implement the extended (or modern) regular expressions as defined in POSIX 1003.2.

Community regular expressions evaluate the string specified in *term* on a character-by-character basis. For example, if you specify 1234:5678 as *term*, the regular expressions see nine discrete characters, including the colon (:), instead of two sets of numbers (1234 and 5678) separated by a colon.

Table 16: Community Attribute Regular Expression Operators

Operator	Match...
{ <i>m,n</i> }	At least <i>m</i> and at most <i>n</i> repetitions of <i>term</i> . Both <i>m</i> and <i>n</i> must be positive integers, and <i>m</i> must be smaller than <i>n</i> .
{ <i>m</i> }	Exactly <i>m</i> repetitions of <i>term</i> . <i>m</i> must be a positive integer.
{ <i>m</i> ,}	<i>m</i> or more repetitions of <i>term</i> . <i>m</i> must be a positive integer.
*	Zero or more repetitions of <i>term</i> . This is equivalent to {0,}.
+	One or more repetitions of <i>term</i> . This is equivalent to {1,}.
?	Zero or one repetition of <i>term</i> . This is equivalent to {0,1}.
	One of the two terms on either side of the pipe.
–	Between a starting and ending range, inclusive.
^	Character at the beginning of a community attribute regular expression. We recommend the use of this operator for the clearest interpretation of your community attribute regular expression. If you do not use this operator, the regular expression 123:456 could also match a route tagged with 5123:456.

Operator	Match...
\$	Character at the end of a community attribute regular expression. We recommend the use of this operator for the clearest interpretation of your community attribute regular expression. If you do not use this operator, the regular expression 123:456 could also match a route tagged with 123:4563.
[]	Set of characters. One character from the set can match. To specify the start and end of a range, use a hyphen (-). To specify a set of characters that do not match, use the caret (^) as the first character after the opening square bracket ([].
()	A group of terms that are enclosed in the parentheses. If enclosed in quotation marks with no intervening space ("()"), indicates a null. Intervening space between the parentheses and the terms is ignored.

Table 17: Examples of Defining Community Attribute Regular Expressions

Community Attribute to Match	Regular Expression	Example Matches
AS number is 56 or 78. Community value is any number.	<code>^((56) (78)):(.*)\$</code>	56:1000 78:65000
AS number is 56. Community value is any number that starts with 2.	<code>^56:(2.*)\$</code>	56:2 56:222 56:234
AS number is any number. Community value is any number that ends with 5, 7, or 9.	<code>^(.*):(.*[579])\$</code>	1234:5 78:2357 34:65009
AS number is 56 or 78. Community value is any number that starts with 2 and ends with 2 through 8.	<code>^((56) (78)):(2.*[2-8])\$</code>	56:22 56:21197 78:2678

Do Not Advertise Communities to Neighbors

By default, communities are sent to BGP peers. To suppress the advertisement of communities to a neighbor, remove all communities. When the result of an export policy is an empty set of communities, the community attribute is not sent. To remove all communities, first define a wildcard set of communities (here, the community is named wild):

```
[edit policy-options]
community wild members "*" : "*" ;
```

Then, in the routing policy statement, specify the community delete action:

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    then community delete wild;
  }
}
```

To suppress a particular community from any AS, define the community as community wild members "*" : *community-value*.

Examples: Configure the Community Attribute

Create a community named dunedin and apply it in a routing policy statement:

```
[edit]
policy-options {
  community dunedin members [56:2379 23:46944];
  policy-statement from-dunedin {
    from community dunedin;
    then {
      metric 2;
      preference 100;
      next policy;
    }
  }
}
```

The above example modifies the metric and preference for routes that contain members of community dunedin only.



NOTE: You cannot set or add a community in a policy whose members use regular expressions or a wildcard.

Delete a particular community from a route, leaving remaining communities untouched:

```
[edit]
policy-options {
  community dunedin members 701:555;
  policy-statement delete-dunedin {
    then {
      community delete dunedin;
    }
  }
}
```

Remove any community from a route with the AS number of 65534 or 65535:

```
[edit]
policy-options {
  community my-as1-transit members [65535:10 65535:11];
  community my-as2-transit members [65534:10 65534:11];
  community my-wild members [65534:* 65535:*];
  policy-statement delete-communities {
    from {
      community [my-as1-transit my-as2-transit];
    }
    then {
      community delete my-wild;
    }
  }
}
```

Match the set of community members 5000, 5010, 5020, 5030, and so on up to 5090:

```
[edit]
policy-options {
  community customers members "^1111:50.0$"
  policy-statement advertise-customers {
    from community customers;
    then accept;
  }
}
```

Reject routes that are longer than /19 in Class A space, /16 in Class B space, and /24 in Class C space:

```
[edit policy-options]
community auckland-accept members 555:1;
policy-statement drop-specific-routes {
  from {
    route-filter 0.0.0.0/1 upto /19 {
      community add auckland-accept
      next policy;
    }
    route-filter 172.16.0.0/2 upto /16 {
      community add auckland-accept
      next policy;
    }
    route-filter 192.168.0.0/3 upto /24 {
      community add auckland-accept
      next policy;
    }
  }
  then reject;
}
```

In the above example, for routes that are not rejected, the tag auckland-accept is added.

Create routing policies to handle peer and customer communities. This example does the following:

Customer routes that match the attributes defined in the lcl20x-low communities, for example, lcl201-low, are accepted and their local preference is changed to 80.

Customer routes that match the attributes defined in the lcl20x-high communities, for example, lcl201-high, are accepted and have their local preference changed to 120.

Internal routes that match the attributes defined in the internal20x communities, for example, internal201, are rejected and not advertised to customers.

Routes received from a peer are assigned a metric of 10 and the community defined in peer201.

Routes that match the attributes defined in the prepend20x-x communities, for example, prepend201-1, prepend201-2, or prepend201-3, are sent to peers and have the AS number 201 prepended the specified number of times.

Routes that match the attributes defined in the peer20x, custpeer20x, and internal20x communities, for example, peer201, custpeer201, or internal201, respectively, are rejected and not advertised to peers.

```
[edit]
policy-options {
  community internal201 members 201:112;
  community internal202 members 202:112;
  community internal203 members 203:112;
  community internal204 members 204:112;
  community internal205 members 205:112;
  community peer201 members 201:555;
  community peer202 members 202:555;
  community peer203 members 203:555;
  community peer204 members 204:555;
  community peer205 members 205:555;
  community custpeer201 members 201:20;
  community custpeer202 members 202:20;
  community custpeer203 members 203:20;
  community custpeer204 members 204:20;
  community custpeer205 members 205:20;
  community prepend201-1 members 201:1;
  community prepend202-1 members 202:1;
  community prepend203-1 members 203:1;
  community prepend204-1 members 204:1;
  community prepend205-1 members 205:1;
  community prepend201-2 members 201:2;
  community prepend202-2 members 202:2;
  community prepend203-2 members 203:2;
  community prepend204-2 members 204:2;
  community prepend205-2 members 205:2;
  community prepend201-3 members 201:3;
  community prepend202-3 members 202:3;
  community prepend203-3 members 203:3;
  community prepend204-3 members 204:3;
  community prepend205-3 members 205:3;
  community lcl201-low members 201:80;
  community lcl202-low members 202:80;
  community lcl203-low members 203:80;
  community lcl204-low members 204:80;
  community lcl205-low members 205:80;
  community lcl 20x-high members "^20 [ 1-5 ] : 120$";
```

```

policy-statement in-customer {
  term term1 {
    from {
      protocol bgp;
      community lcl 20x-high;
    }
    then {
      local-preference 80;
      accept;
    }
  }
  term term2 {
    from {
      protocol bgp;
      community [lcl201-high lcl202-high lcl203-high lcl204-high lcl205-high];
    }
    then local-preference 120;
  }
  then next policy;
}
policy-statement out-customer {
  term term1 {
    from {
      protocol bgp;
      community [internal201 internal202 internal203 internal204
        internal205];
    }
    then reject;
  }
  then next policy;
}
policy-statement in-peer {
  from protocol bgp;
  then {
    metric 10;
    community set peer201;
  }
}
policy-statement out-peer {
  term term1{
    from {
      protocol bgp;
      community [prepend201-1 prepend202-1 prepend203-1 prepend204-1
        prepend205-1];
    }
    then as-path-prepend 201;
  }
  term term2 {
    from {
      protocol bgp;
      community [prepend201-2 prepend202-2 prepend203-2 prepend204-2
        prepend205-2];
    }
    then as-path-prepend "201 201";
  }
  term term3 {
    from {

```

```

        protocol bgp;
        community [prepend201-3 prepend202-3 prepend203-3
        prepend204-3 prepend205-3];
    }
    then as-path-prepend "201 201 201";
}
term term4 {
    from {
        protocol bgp;
        community [peer201 peer202 peer203 peer204 peer205 custpeer201
        custpeer202 custpeer203 custpeer204 custpeer205 internal201
        internal202 internal203 internal204 internal205];
    }
    then reject;
}
then next policy;
}
}

```

Configure the Extended Communities Attribute

To configure extended communities, include the community statement:

```

community name {
    invert-match;
    members [ community-ids ];
}

```

You can include this statement at the following hierarchy levels:

```
[edit policy-options]
```

```
[edit logical-routers logical-router-name policy-options]
```

To include the community in a routing policy, include the community condition in the from statement:

```

[edit policy-options]
community name members [ community-ids ];
policy-statement policy-name {
    term term-name {
        from {
            community name;
        }
    }
}
}

```

name identifies one or more routers in the BGP extended community.

community-ids identifies the type of extended community in the following format:

```
type:administrator:assigned-number
```

type is the type of extended community and can be either a bandwidth, target, origin, or domain-id community. The bandwidth community sets up the bandwidth extended community. The target community identifies the destination to which the route is going. The origin community identifies where the route originated. The domain-id community identifies the Open Shortest Path First (OSPF) domain from which the route originated.

administrator is the administrator. It is either an AS number or an Internet Protocol version 4 (IPv4) address prefix, depending on the type of extended community.

assigned-number identifies the local provider.

For more information about BGP extended communities, see Internet draft *draft-ramachandra-bgp-ext-communities-version.txt*, *BGP Extended Communities Attribute*. For information about the community attribute, see “Configure the Community Attribute” on page 97.



NOTE: Regular expressions are not supported for extended communities.

Examples: Configure the Extended Communities Attribute

Configure a target community with an administrative field of 10458 and an assigned number of 20:

```
[edit]
policy-options {
  community test-a members [target:10458:20];
}
```

Configure a target community with an administrative field of 1.1.1.1 and an assigned number of 20:

```
[edit]
policy-options {
  community test-a members [target:1.1.1.1:20];
}
```

Configure an origin community with an administrative field of 1.1.1.1 and an assigned number of 20:

```
[edit]
policy-options {
  community test-a members [origin:1.1.1.1:20];
}
```

Invert Community Matches

To invert the results of the community expression matching, include the `invert-match` statement:

```
invert-match;
```

You can include this statement at the following hierarchy levels:

```
[edit policy-options community name]
```

```
[edit logical-routers logical-router-name policy-options community name]
```

Configure Link Bandwidth

To configure the link bandwidth extended communities attribute, include the `bandwidth` statement:

```
bandwidth:as:bandwidth;
```

You can include this statement at the following hierarchy levels:

```
[edit policy-options community name members]
```

```
[edit logical-routers logical-router-name policy-options community name members]
```

Specifying link bandwidth allows you to distribute traffic unequally among different BGP paths.



CAUTION: The link bandwidth attribute does not work concurrently with per-packet load-balancing.

How Communities Are Evaluated

The policy framework software evaluates communities as follows:

Each route is evaluated against each named community in a routing policy from statement. If a route matches one of the named communities in the from statement, the evaluation of the current term continues. If a route does not match, the evaluation of the current term ends.

The route is evaluated against each member of a named community. The evaluation of all members must be successful for the named community evaluation to be successful.

Each member in a named community is either a literal community value or a regular expression. Each member is evaluated against each community associated with the route. (Communities are an unordered property of a route. For example, 1:2 3:4 is the same as 3:4 1:2.) Only one community from the route is required to match for the member evaluation to be successful.

Community regular expressions are evaluated on a character-by-character basis. For example, if a route contains community 1234:5678, the regular expressions see nine discrete characters, including the colon (:), instead of two sets of numbers (1234 and 5678) separated by a colon. For example:

```
[edit]
policy-options {
  policy-statement one {
    from {
      community [comm-one comm-two];
    }
  }
  community members comm-one [ 1:2 "^4:(5|6)$" ];
  community members comm-two [ 7:8 9:10 ];
}
```

To match routing policy one, the route must match either comm-one or comm-two.

To match comm-one, the route must have a community that matches 1:2 and a community that matches 4:5 or 4:6.

To match comm-two, the route must have a community that matches 7:8 and a community that matches 9:10.

Configure Prefix Lists

A *prefix list* is a named list of IP addresses. You can specify an exact match with incoming routes and apply a common action to all matching prefixes in the list.



NOTE: Because the configuration of prefix lists includes setting up prefixes and prefix lengths, we strongly recommend that you have a thorough understanding of IP addressing, including supernetting, before proceeding with the configuration.

This section includes the following information:

Prefix List and Route List Differences on page 109

Define Prefix Lists on page 109

How a Prefix List Is Evaluated on page 110

Example: Configure a Prefix List on page 111

Prefix List and Route List Differences

A prefix list functions similarly to a route list that contains multiple instances of the exact match type only. The similarities between these two extended match conditions which are summarized in Table 18.

Table 18: Prefix List and Route List Differences

Feature	Prefix List	Route Lists
Match types	Does not support match types. The specified prefixes must be matched exactly.	Supports several match types. For more information, see Table 19 on page 113.
Action	Can specify action in a then statement only. These actions are applied to all prefixes that match the term.	Can specify action that is applied to a particular prefix in a route-filter match condition in a from statement, or to all prefixes in the list using a then statement.

Define Prefix Lists

You can create a named prefix list and include it in a routing policy with the prefix-list match condition (described in Table 9 on page 44).

To define a prefix list, include the prefix-list statement:

```
prefix-list name {
  apply-path path;
  ip-addresses;
}
```

You can include this statement at the following hierarchy levels:

```
[edit policy-options]
```

```
[edit logical-routers logical-router-name policy-options]
```

You can use the apply-path statement to include all prefixes pointed to by a defined path, or you can specify one or more addresses, or both.

To include a prefix list in a routing policy, specify the prefix-list match condition in the from statement:

```
[edit]
policy-options {
  policy-statement policy-name {
    term term-name {
      from {
        prefix-list name;
      }
      then actions;
    }
  }
}
```

name identifies the prefix list. It can contain letters, numbers, and hyphens (-) and can be up to 255 characters long. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

ip-addresses are the IPv4 or Internet Protocol version 6 (IPv6) prefixes specified as *prefix/prefix-length*. If you omit *prefix-length* for an IPv4 prefix, the default is /32. If you omit *prefix-length* for an IPv6 prefix, the default is /128. Prefixes specified in a from statement must be either all IPv4 addresses or all IPv6 addresses.



NOTE: You cannot apply actions to individual prefixes in the list.

You can specify the same prefix list in the from statement of multiple routing policies or firewall filters. For information about firewall filters, see “Firewall Filters” on page 147.

Use the *apply-path* statement to configure a prefix list comprising of all IP prefixes pointed to by a defined path. This eliminates most of the effort required to maintain a group prefix list.

The path consists of elements separated by spaces. Each element matches a configuration keyword or an identifier, and you can use wildcards to match more than one identifier. Wildcards must be enclosed in angle brackets, for example, `< * >`.



NOTE: When you use *apply-path* to define a prefix list, the prefix list can be used in a firewall filter only. It cannot be used in a policy statement.

For examples of configuring a prefix list, see “Example: Configure a Prefix List” on page 111, and for examples of configuring a firewall filter, see “Configure Firewall Filters” on page 155.

How a Prefix List Is Evaluated

During prefix list evaluation, the policy framework software performs a *longest-match lookup*, which means that the software searches for the prefix in the list with the longest length. The order in which you specify the prefixes, from top to bottom, does not matter. The software then compares a route’s source address to the longest prefix.

If a match occurs, the evaluation of the current term continues. If a match does not occur, the evaluation of the current term ends.



NOTE: If you specify multiple prefixes in the prefix list, only one prefix must match for a match to occur. The prefix list matching is effectively a logical OR operation.

Example: Configure a Prefix List

The following example accepts and rejects traffic from sites specified using prefix lists:

```
[edit]
policy-options {
  policy-statement prefix-list-policy {
    term ok-sites {
      from {
        prefix-list known-ok-sites
      }
      then accept;
    }
    term reject-bcasts {
      from {
        prefix-list known-dir-bcast-sites
      }
      then reject;
    }
  }
}
```

```
[edit]
policy-options {
  prefix-list known-ok-sites {
    172.16.0.3;
    10.10.0.0/16;
    192.168.12.0/24;
  }
}
```

```
[edit]
policy-options {
  prefix-list known-dir-bcast-sites {
    10.3.4.6;
    10.2.0.0/16;
    192.168.1.0/24;
  }
}
```

Configure Route Lists

A *route list* is a collection of destination prefixes. When specifying a prefix, you can specify an exact match with a particular route or a less precise match. You can configure either a common action that applies to the entire list or an action associated with each prefix.



NOTE: Because the configuration of route lists includes setting up prefixes and prefix lengths, we strongly recommend that you have a thorough understanding of IP addressing, including supernetting, before proceeding with the configuration.

It is also important to understand how a route list is evaluated, particularly if the route list includes multiple route-filter options in a from statement. We strongly recommend that you read “How a Route List Is Evaluated” on page 114 before proceeding with the configuration. Not fully understanding the evaluation process could result in faulty configuration and unexpected results.

This section discusses the following topics:

Define Route Lists on page 112

How a Route List Is Evaluated on page 114

Examples: Configure Route Lists on page 116

Define Route Lists

To specify a route list, include one or more route-filter or source-address-filter options in the from statement of the policy-statement:

```
[edit]
policy-options {
  policy-statement policy-name {
    term term-name {
      from {
        route-filter destination-prefix match-type <actions>;
        source-address-filter destination-prefix match-type <actions>;
      }
      then actions;
    }
  }
}
```

The route-filter option is typically used to match prefixes of any type except for multicast source addresses.

The source-address-filter option is typically used to match multicast source addresses in multiprotocol BGP (MBGP) and Multicast Source Discovery Protocol (MSDP) environments.

destination-prefix is the IPv4 or IPv6 prefix specified as *prefix/prefix-length*. If you omit *prefix-length* for an IPv4 prefix, the default is /32. If you omit *prefix-length* for an IPv6 prefix, the default is /128. Prefixes specified in a from statement must be either all IPv4 addresses or all IPv6 addresses.

match-type is the type of match to apply to the destination prefix. It can be one of the match types listed in Table 19. For examples of the match types and the results when presented with various routes, see Table 20.

actions is the action to take if the destination prefix matches. It can be one or more of the actions listed in Table 10 on page 49 and Table 11 on page 49.

In route lists, you can specify actions in two ways:

In the route-filter or source-address-filter option—These actions are taken immediately after a match occurs, and the then statement is not evaluated.

In the then statement—These actions are taken after a match occurs and if an action is not specified in the route-filter or source-address-filter option.

The upto and prefix-length-range match types are similar in that both specify the most significant bits and provide a range of prefix lengths that can match. The difference is that upto allows you to specify an upper limit only for the prefix length range, while prefix-length-range allows you to specify both lower and upper limits.

For more examples of these route list match types, see “Examples: Configure Route Lists” on page 116.

Table 19: Route List Match Types

Match Type	Match If ...
exact	The route shares the same most-significant bits (described by <i>prefix-length</i>), and <i>prefix-length</i> is equal to the route’s prefix length.
longer	The route shares the same most-significant bits (described by <i>prefix-length</i>), and <i>prefix-length</i> is greater than the route’s prefix length.
orlonger	The route shares the same most-significant bits (described by <i>prefix-length</i>), and <i>prefix-length</i> is equal to or greater than the route’s prefix length.
prefix-length-range <i>prefix-length2</i> – <i>prefix-length3</i>	The route shares the same most-significant bits (described by <i>prefix-length</i>), and the route’s prefix length falls between <i>prefix-length2</i> and <i>prefix-length3</i> , inclusive.
through <i>destination-prefix</i>	All the following are true: The route shares the same most-significant bits (described by <i>prefix-length</i>) of the first destination prefix. The route shares the same most-significant bits (described by <i>prefix-length</i>) of the second destination prefix for the number of bits in the prefix length. The number of bits in the route’s prefix length is less than or equal to the number of bits in the second prefix. You do not use the through match type in most routing policy configurations. (For an example, see “Examples: Configure Route Lists” on page 116.)
upto <i>prefix-length2</i>	The route shares the same most-significant bits (described by <i>prefix-length</i>) and the route’s prefix length falls between <i>prefix-length</i> and <i>prefix-length2</i> .

Table 20: Match Type Examples

Prefix	192.168/16 exact	192.168/16 longer	192.168/16 orlonger	192.168/16 upto /24	192.168/16 through 192.168.16/20	192.168/16 prefix-length- range /18-/20
10.0.0.0/8						
192.168.0.0/16	Match		Match	Match	Match	
192.168.0.0/17		Match	Match	Match	Match	
192.168.0.0/18		Match	Match	Match	Match	Match
192.168.0.0/19		Match	Match	Match	Match	Match
192.168.4.0/24		Match	Match	Match		
192.168.5.4/30		Match	Match			
192.168.12.4/30		Match	Match			
192.168.12.128/32		Match	Match			
192.168.16.0/20		Match	Match	Match	Match	Match
192.168.192.0/18		Match	Match	Match		Match
192.168.224.0/19		Match	Match	Match		Match
10.169.1.0/24						
10.170.0.0/16						

How a Route List Is Evaluated

During route list evaluation, the policy framework software compares each route's source address with the destination prefixes in the route list. The evaluation occurs in two steps:

1. The policy framework software performs a *longest-match lookup*, which means that the software searches for the prefix in the list with the longest length.

The longest-match lookup considers the prefix and prefix length only and not the match type. The following sample route list illustrates this point:

```
from {
    route-filter 192.168.0.0/14 upto /24 reject;
    route-filter 192.168.0.0/15 exact;
}
then accept;
```

The longest match is the second route-filter, 192.168.0.0/15, which is based on prefix and prefix length only.

2. Once an incoming route matches a prefix (longest first), the following occurs:

The route filter stops evaluating other prefixes, even if the match type fails.

The software examines the match type and action associated with that prefix.

In Step 1, if route 192.168.1.0/24 were evaluated, it would fail to match. It matches the longest prefix of 192.168.0.0/15, but it does not match exact. The route filter is finished because it matched a prefix, but the result is a failed match because the match type failed.

If a match occurs, the action specified with the prefix is taken. If an action is not specified with the prefix, the action in the then statement is taken. If neither action is specified, the software evaluates the next term or routing policy, if present, or takes the accept or reject action specified by the default policy. For more information about the default routing policies, see “Default Routing Policies and Actions” on page 21.



NOTE: If you specify multiple prefixes in the route list, only one prefix needs to match for a match to occur. The route list matching is effectively a logical OR operation.

If a match does not occur, the software evaluates the next term or routing policy, if present, or takes the accept or reject action specified by the default policy.

For example, compare the prefix 192.168.254.0/24 against the following route list:

```
route-filter 192.168.0.0/16 orlonger;
route-filter 192.168.254.0/23 exact;
```

The prefix 192.168.254.0/23 is determined to be the longest prefix. When evaluating 192.168.254.0/24 against the longest prefix, a match occurs (192.168.254.0/24 is a subset of 192.168.254.0/23). Because of the match between 192.168.254.0/24 and the longest prefix, the evaluation continues. However, when evaluating the match type, a match does not occur between 192.168.254.0/24 and 192.168.254.0/23 exact. The software concludes that the term does not match and goes on to the next term or routing policy, if present, or takes the accept or reject action specified by the default policy.

The following section describes the following topics:

How Prefix Order Affects Route List Evaluation on page 115

Common Configuration Problem with the Longest-Match Lookup on page 116

How Prefix Order Affects Route List Evaluation

The order in which the prefixes are specified (from top to bottom) typically does not matter, because the policy framework software scans the route list looking for the longest prefix during evaluation. An exception to this rule is when you use the same destination prefix multiple times in a list. In this case, the order of the prefixes is important, because the list of identical prefixes is scanned from top to bottom, and the first match type that matches the route applies.

In the following example, different match types are specified for the same prefix. The route 0.0.0.0/0 would be rejected, the route 0.0.0.0/8 would be marked with next-hop self, and the route 0.0.0.0/25 would be rejected.

```
route-filter 0.0.0.0/0 upto /7 reject;
route-filter 0.0.0.0/0 upto /24 next-hop self;
route-filter 0.0.0.0/0 orlonger reject;
```

Common Configuration Problem with the Longest-Match Lookup

A common problem when defining a route list is including a shorter prefix that you want to match with a longer, similar prefix in the same list. For example, imagine that the prefix 192.168.254.0/24 is compared against the following route list:

```
route-filter 192.168.0.0/16 orlonger;
route-filter 192.168.254.0/23 exact;
```

Because the policy-framework software performs longest-match lookup, the prefix 192.168.254.0/23 is determined to be the longest prefix. An exact match does not occur between 192.168.254.0/24 and 192.168.254.0/23 exact. The software concludes that the term does not match and goes on to the next term or routing policy, if present, or takes the accept or reject action specified by the default policy. (For more information about the default routing policies, see “Default Routing Policies and Actions” on page 21.) The shorter prefix 192.168.0.0/16 orlonger that you wanted to match is inadvertently ignored.

One solution to this problem is to remove the prefix 192.168.0.0/16 orlonger from the route list in this term and move it to a previous term where it is the only prefix or the longest prefix in the list.

Examples: Configure Route Lists

The examples in this section show only fragments of routing policies. Normally, you would combine these fragments with other terms or routing policies.

In all examples, remember that the following actions apply to nonmatching routes:

Evaluate next term, if present.

Evaluate next policy, if present.

Take the accept or reject action specified by the default policy. For more information about the default routing policies, see “Default Routing Policies and Actions” on page 21.

Reject routes with a destination prefix of 0.0.0.0 and a mask length from 0 through 8, and accept all other routes:

```
[edit]
policy-options {
  policy-statement from-hall2 {
    term 1 {
      from {
        route-filter 0.0.0.0/0 upto /8 reject;
      }
    }
    then accept;
  }
}
```

Reject routes with a mask of /8 and greater (that is, /8, /9, /10, and so on) that have the first 8 bits set to 0 and accept routes less than 8 bits in length:

```
[edit]
policy-options {
  policy-statement from-hall3 {
    term term1 {
      from {
        route-filter 0/0 upto /7 accept;
        route-filter 0/8 orlonger;
      }
      then reject;
    }
  }
}
```

Reject routes with the destination prefix of 192.168.10/24 and a mask between /26 and /29 and accept all other routes:

```
[edit]
policy-options {
  policy-statement from-customer-a {
    term term1 {
      from {
        route-filter 192.168.10/24 prefix-length-range /26-/29 reject;
        route-filter 0/0;
      }
      then accept;
    }
  }
}
```

Reject a range of routes from specific hosts, and accept all other routes:

```
[edit]
policy-options {
  policy-statement hosts-only {
    from {
      route-filter 10.125.0.0/16 upto /31 reject;
      route-filter 0/0;
    }
    then accept;
  }
}
```

You do not use the through match type in most routing policy configurations. You should think of through as a tool to group a contiguous set of exact matches. For example, instead of specifying four exact matches:

```
from route-filter 0.0.0.0/1 exact
from route-filter 0.0.0.0/2 exact
from route-filter 0.0.0.0/3 exact
from route-filter 0.0.0.0/4 exact
```

You could represent them with the following single match:

```
from route-filter 0.0.0.0/1 through 0.0.0.0/4
```

Explicitly accept a limited set of prefixes (in the first term) and reject all others (in the second term):

```
[edit policy-options]
policy-statement internet-in {
  term 1 {
    from {
      route-filter 192.168.231.0/24 exact accept;
      route-filter 192.168.244.0/24 exact accept;
      route-filter 192.168.198.0/24 exact accept;
      route-filter 192.168.160.0/24 exact accept;
      route-filter 192.168.59.0/24 exact accept;
    }
  }
  term 2 {
    then {
      reject;
    }
  }
}
```

Reject a few groups of prefixes, then accept the remaining prefixes:

```
[edit policy-options]
policy-statement drop-routes {
  term 1{
    from {
      route-filter default exact reject;      # first, reject a number of prefixes:
      route-filter 0.0.0.0/8 orlonger reject; # reject 0.0.0.0/0 exact
      route-filter 10.0.0.0/8 orlonger reject; # reject prefix 0, mask /8 or longer
      route-filter 10.105.0.0/16 exact {      # reject loopback addresses
        as-path-prepend "1 2 3";           # accept 10.105.0.0/16
        accept;
      }
      route-filter 192.0.2.0/24 orlonger reject; # reject test network packets
      route-filter 224.0.0.0/3 orlonger reject; # reject multicast and higher
      route-filter 0.0.0.0/0 upto /24 accept;  # accept everything up to /24
      route-filter 0.0.0.0/0 orlonger accept;  # accept everything else
    }
  }
}
```

Reject all prefixes longer than 24 bits. You would install this routing policy in a sequence of routing policies in an export statement. The first term in this filter passes on all routes with a prefix length of up to 24 bits. The second, unnamed term rejects everything else.

```
[edit policy-options]
policy-statement 24bit-filter {
  term acl20 {
    from {
      route-filter 0.0.0.0/0 upto /24;
    }
    then next policy;
  }
  then reject;
}
```

If, in this example, you were to specify `route-filter 0.0.0.0/0 upto /24 accept`, matching prefixes would be accepted immediately and the next routing policy in the export statement would never get evaluated.

If you were to include the `then reject` statement in the term `acl20`, prefixes greater than 24 bits would never get rejected because the policy framework software, when evaluating the term, would move on to evaluating the next statement before reaching the `then reject` statement.

Configure a routing policy for rejecting Protocol Independent Multicast (PIM) multicast traffic joins for a source destination prefix from a neighbor:

```
[edit]
policy-options {
  policy-statement join-filter {
    from {
      neighbor 10.14.12.20;
      source-address-filter 10.83.0.0/16 orlonger;
    }
    then reject;
  }
}
```

Configure a routing policy for rejecting PIM traffic for a source destination prefix from an interface:

```
[edit]
policy-options {
  policy-statement join-filter {
    from {
      interface so-1/0/0.0;
      source-address-filter 10.83.0.0/16 orlonger;
    }
    then reject;
  }
}
```

The following routing policy qualifiers apply to PIM:

interface—Interface over which a join is received

neighbor—Source from which a join originates

route-filter—Group address

source-address-filter—Source address for which to reject a join

For more information about importing a PIM join filter in a PIM protocol definition, see the *JUNOS Internet Software Multicast Protocols Configuration Guide*.

Configure Subroutines

You can use a routing policy called from another routing policy as a match condition. This process makes the called policy a *subroutine*.

This section describes the following task for configuring subroutines and provides the following example:

Define Subroutines on page 121

Example: Configure a Subroutine on page 124

Define Subroutines

To configure a subroutine in a routing policy to be called from another routing policy, create the subroutine and specify its name using the policy match condition (described in Table 9 on page 44) in the from or to statement of another routing policy:

```
[edit]
policy-options {
  policy-statement subroutine-policy-name {
    term term-name {
      from {
        match-conditions;
        route-filter destination-prefix match-type <actions>;
        source-address-filter destination-prefix match-type <actions>;
        prefix-list name;
      }
      to {
        match-conditions;
      }
      then actions;
    }
  }
}

policy-options {
  policy-statement policy-name {
    term term-name {
      from {
        policy subroutine-policy-name;
      }
      to {
        policy subroutine-policy-name;
      }
      then actions;
    }
  }
}
```



NOTE: Do not evaluate a routing policy within itself. If you attempt to do so, no prefixes will ever match the routing policy.

The action specified in a subroutine is used to provide a match condition to the calling policy. If the subroutine specifies an action of accept, the calling policy considers the route to be a match. If the subroutine specifies an action of reject, the calling policy considers the route not to match. If the subroutine specifies an action that is meant to manipulate the route characteristics, the changes are made. For more details about the subroutine evaluation, see “How a Routing Policy Subroutine Is Evaluated” on page 33.

Termination Actions

A subroutine with particular statements can behave differently from a routing policy that contains the same statements. With a subroutine, you must remember that the possible termination actions of accept or reject specified by the subroutine or the default policy can greatly affect the expected results. (For more information about the default routing policies, see “Default Routing Policies and Actions” on page 21.)

In particular, you must consider what happens if a match does not occur with routes specified in a subroutine and if the default policy action that is taken is the action that you expect and want.

For example, imagine that you are a network administrator at an Internet service provider (ISP) that provides service to Customer A. You have configured several routing policies for the different classes of neighbors that Customer A presents on various links. To save time maintaining the routing policies for Customer A, you have configured a subroutine that identifies their routes and various routing policies that call the subroutine, as shown below:

```
[edit]
policy-options {
  policy-statement customer-a-subroutine {
    from {
      route-filter 10.1/16 exact;
      route-filter 10.5/16 exact;
      route-filter 192.168.10/24 exact;
    }
    then accept;
  }
}
policy-options {
  policy-statement send-customer-a-default {
    from {
      policy customer-a-subroutine;
    }
    then {
      set metric 500;
      accept;
    }
  }
}
policy-options {
  policy-statement send-customer-a-primary {
    from {
      policy customer-a-subroutine;
    }
    then {
      set metric 100;
      accept;
    }
  }
}
```

```

policy-options {
  policy-statement send-customer-a-secondary {
    from {
      policy customer-a-subroutine;
    }
    then {
      set metric 200;
      accept;
    }
  }
}
protocols {
  bgp {
    group customer-a {
      export send-customer-a-default;
      neighbor 10.1.1.1;
      neighbor 10.1.2.1;
      neighbor 10.1.3.1 {
        export send-customer-a-primary;
      }
      neighbor 10.1.4.1 {
        export send-customer-a-secondary;
      }
    }
  }
}

```

The following results occur with this configuration:

The group-level export statement resets the metric to 500 when advertising all BGP routes to neighbors 1.1.1.1 and 1.1.2.1 rather than just the routes that match the subroutine route filters.

The neighbor-level export statements reset the metric to 100 and 200 when advertising all BGP routes to neighbors 1.1.3.1 and 1.1.4.1, respectively, rather than just the BGP routes that match the subroutine route filters.

These unexpected results occur because the subroutine policy does not specify a termination action for routes that do not match the route filter and therefore, the default BGP export policy of accepting all BGP routes is taken.

If the statements included in this particular subroutine had been contained within the calling policies themselves, only the desired routes would have their metrics reset.

This example illustrates the differences between routing policies and subroutines and the importance of the termination action in a subroutine. Here, the default BGP export policy action for the subroutine was not carefully considered. A solution to this particular example is to add one more term to the subroutine that rejects all other routes that do not match the route filters:

```
[edit]
policy-options {
  policy-statement customer-a-subroutine {
    term accept-exact {
      from {
        route-filter 10.1/16 exact;
        route-filter 10.5/16 exact;
        route-filter 192.168.10/24 exact;
      }
      then accept;
    }
    term reject-others {
      then reject;
    }
  }
}
```

Termination action strategies for subroutines in general include the following:

- Depend upon the default policy action to handle all other routes.

- Add a term that accepts all other routes. (Also see “Side Effects of Omitting the “from” Statement from an Export Policy” on page 63.)

- Add a term that rejects all other routes.

The option that you choose depends upon what you want to achieve with your subroutine. Plan your subroutines carefully.

Example: Configure a Subroutine

Create the subroutine `is-customer` and call it from the routing policies `export-customer` and `import-customer`. In `import-customer`, the action is taken only on routes that match the route filters defined in `is-customer`.

```
[edit]
policy-options {
  policy-statement is-customer {
    term match-customer {
      from {
        route-filter 10.100.1.0/24 exact;
        route-filter 10.186.100.0/24 exact;
      }
      then accept;
    }
    term drop-others {
      then reject;
    }
  }
}
```

```
policy-statement export-customer {  
  from policy is-customer;  
  then accept;  
}  
policy-statement import-customer {  
  from {  
    protocol bgp;  
    policy is-customer;  
  }  
  then {  
    local-preference 10;  
    accept;  
  }  
}  
}
```

