Policy User Guide

Version 22.7.1, 25 July 2022

Registered Address	Support	Sales
26, Kingston Terrace, Princeton, New Jersey 08540, United States		
		+91 80 4850 5445
http://www.rtbrick.com	support@rtbrick.com	sales@rtbrick.com

Table of Contents

1. Introduction . . . 3
1.1. Policy Components . . . 3
1.1.1. Policy Repository . . . 3
1.1.2. Command Processing Module . . . 3
1.1.3. Policy Server . . . 4
1.1.4. Policy Client . . . 4
1.1.5. Tables and Subscriptions . . . 5
1.2. Policy Building Blocks . . . 5
1.2.1. Statements . . . 6
1.2.1.1. Ordinals . . . 6
1.2.1.2. Match Rules . . . 6
1.2.1.3. Action Rules . . . 7
1.2.2. Lists . . . 7
1.2.3. Conditions . . . 7
1.2.4. Attachment Points . . . 8
1.3. Policy Behaviour . . . 8
1.4. Supported Platforms . . . 9
2. Policy Configuration . . . 10
2.1. Configuration Hierarchy . . . 10
2.2. Configuration Syntax and Commands . . . 10
2.2.1. Configuring Policy Statements . . . 10
2.2.1.1. Configuring Match Rules . . . 11
2.2.1.2. Configuring Action Rules . . . 14
2.2.2. Configuring Policy Lists . . . 17
2.2.3. Configuring Policy Conditions . . . 19
2.2.3.1. Configuring Table Match . . . 19
2.2.3.2. Configuring Attribute Match . . . 20
2.2.4. Attaching Policies . . . 21
2.2.4.1. BGP Attachment Points . . . 21
2.2.4.2. IS-IS Attachment Points . . . 21
2.2.4.3. IGMP Attachment Points . . . 21
2.3. Sample Configurations . . . 22
3. Operational Commands . . . 26
3.1. Policy Test . . . 26

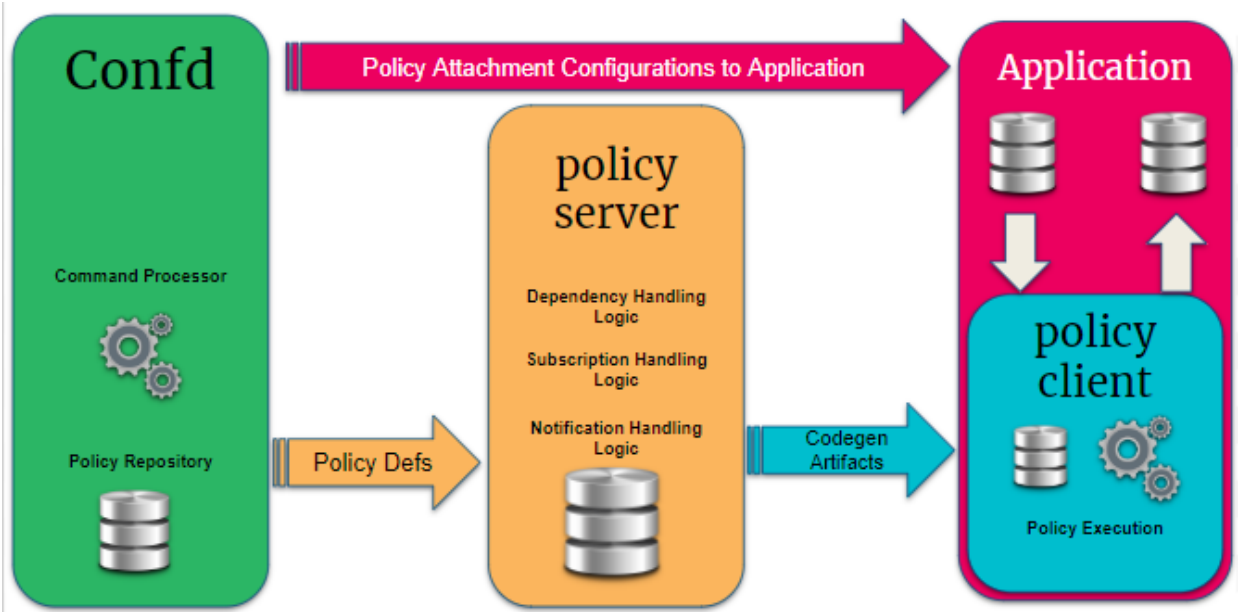
1. Introduction

Policies are set of rules that allow to control and modify the behaviour of routing protocols such as BGP and IS-IS. The policy framework is generic, it serves multiple purposes and applications. Policies are first created using a common policy configuration, and then applied by attaching them to an application like a protocol.

1.1. Policy Components

In RtBrick Full Stack, the policy implementation consists of 4 sub-components:

- Policy Repository
- Command Processing Module
- Policy Server, the policy generation and relationship management component
- Policy Client, the policy enforcement component



1.1.1. Policy Repository

The policy repository contains all tables related to policy and associated list of match criteria.

1.1.2. Command Processing Module

The command processing module is part of the configuration daemon (confd), and it handles user interaction with the policy module. This is the back-end of the Command Line Interface (CLI) and JSON configuration that support the policy configurations.

This module maps the user-defined configuration into the back-end policy object,

which is used by the execution engine (after verification), and it ensures that the policy can be correctly executed. This module relays the user intent via related BDS tables to the policy server.

1.1.3. Policy Server

The policy server is a server component that manages all policy rules in the various policy tables and also code generation of the policies.

The following are functionalities of the policy server:

- Parses the objects in the policy tables, and is an execution engine that generates the code to build the policy rules for evaluation, the relationship between various objects, and relays the intent to the evaluation engine.
- Maintains relationships between various policy constructs such as policy statements, rules, ordinals and lists.
- Tracks the attachment points so that when policies are modified, the appropriate clients are notified with the relevant new policies.
- Flattens the various relationships and generates a notification table that the clients subscribe to obtain notification based on specific interest groups.
- Uses dependency table relationships to generate jobs to trigger code generation for various policy components.
- On code generation, the policy server updates a notification table that maintains the mapping between the policy server and the client interest groups. The notification table is a single point for the dissemination of information so that it can generate notifications for clients depending on their subscriptions for policy of interest.
- Policy server notification is generated for the policy clients. A notification is received from the notification table with metadata information that notifies the client if this is a new version of the policy or the original version of the policy. The client uses this information to enforce the policy evaluation and to decide on the version of the policy rule that should be used.

1.1.4. Policy Client

The policy client is a shared library component that a client daemon like BGP, IS-IS, OSPF etc. links to. This is the component that performs policy enforcement. It performs the following tasks:

- Links with client daemons like BGP, IS-IS, OSPF.
- Contains a listener that gets notifications on the availability of a new policy rule that is generated by the policy server.
- Evaluates the compiled rule and if there are any listeners/ interests, then notifies the components within the client daemon.

- Evaluates any policy configurations on the client daemon and invokes policy processing in response.

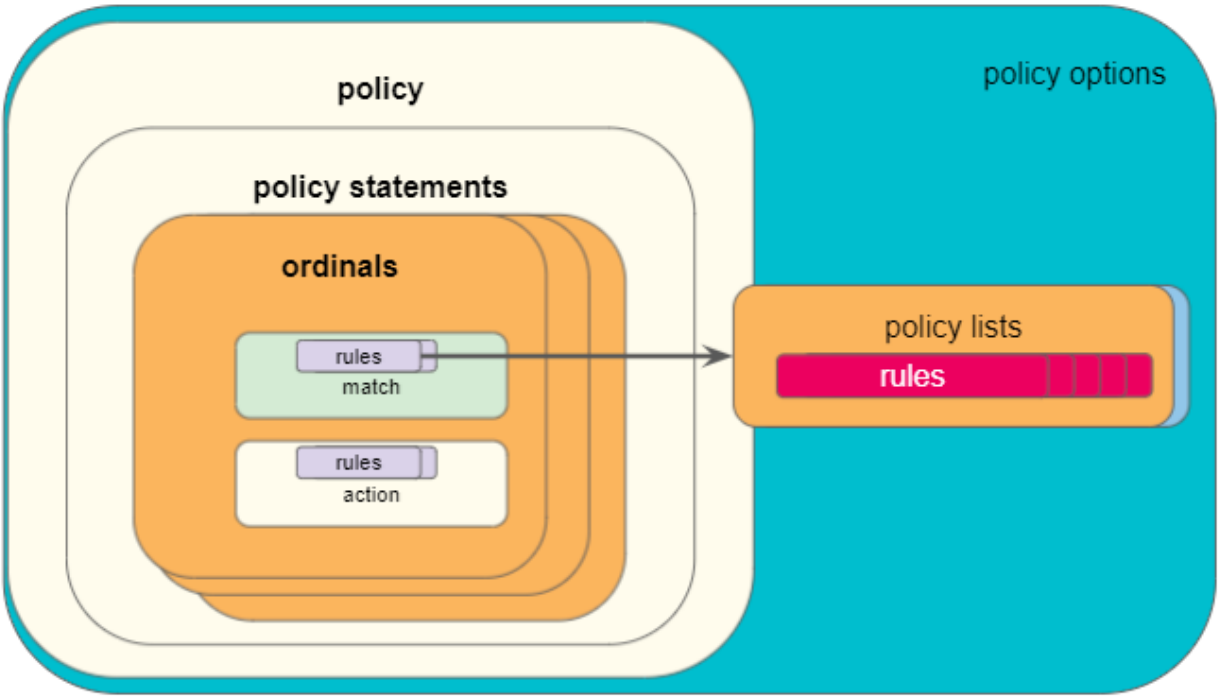
1.1.5. Tables and Subscriptions

The table below shows the various tables and their sharing across various policy components.

Confd	global.policy.list.config global.policy.list.entry.config global.policy.match.rules.config global.policy.statement.config global.policy.ordinal.config global.policy.mapping.list global.policy.mapping.rules	Policy Statement is composed of one or more policy terms. Each term has a match action criteria. In the match and action criteria either a single element or a list of elements are compared and actions are taken. The actions include accept, deny, flow-control etc.
policy.server	global.policy.dependency global.<bds_name>.policy.subscription global.<bds_name>.policy.notification	Policy Server subscribes to all the tables from confd and creates tables that track policy-entry and dependency and notifies clients after code generation.
policy.client	global.<bds_name>.policy.shared.object.cache global.<bds_name>.policy.subscription global.<bds_name>.policy.context	Subscribes to code generation notifications, application context and maintains cache of subscribed .so

1.2. Policy Building Blocks

The figure below shows the building blocks of a policy.



1.2.1. Statements

A policy is defined by a policy statement. A policy statement is a compound block of policy definition that consists of one or more set of rules called ordinals. A policy statement is exercised in the order defined. The statement name is a globally unique string that is used to identify the policy, and used by the applications.

1.2.1.1. Ordinals

A policy ordinal is the smallest block to represent a user policy intent and consists of rules for match and action blocks. The match blocks can either define single independent elements like AS path, IP prefix, IP addresses, community, etc., or a list of the elements maintained in a different table.

- An ordinal must be a unique number within the scope of a statement which determines the order of the term execution within a policy statement.
- If no ordinal exist or configured, and if the policy is used, then all routes or objects will be denied.
- A match logic is defined per ordinal. In case of multiple match rules, it defines if all rules (and), or any of the rules (or) have to match.

1.2.1.2. Match Rules

Match rules define criteria to evaluate and select routes or other objects to which the policy is applied. One or more match rules compose a match block.

- If the match block results in a successful match ("true"), the corresponding action block will executed.

- If the match block result is unsuccessful ("false"), the action block will not be executed, and the next ordinal will be processed.
- If there is no match block, the action block will be executed.

In case of multiple match rules, the behaviour depends on the configured match logic:

- If the match logic is or, the match block result will be succcessful ("true") if any one rule matches. Otherwise, by default it is "false".
- If the match logic is and, the match block result will be successful ("true") if all rules match. Otherwise it is "false".

A match rule can refer to a single discrete value, or a list. A policy list is configured separately and referenced from the policy statement. In case of a list match, the behaviour is as follows:

- If any of the list entries matches the configured value, the match block result will be succcessful ("true"). Otherwise it is "false".
- If the list is defined but empty, the match rule result will be unsuccessful ("false").

1.2.1.3. Action Rules

Action rules define operations like permit or deny, or flow control commands like goto-next-ordinal. The action block will be executed if there is a successful match. Otherwise, the next ordinal is processed. The action block can contain one or multiple action rules. In case of multiple action rules, all actions will be applied.

The action block is optional. The implicit default action is deny.

1.2.2. Lists

A policy list is a list of values that can be referenced by a match rule in a policy statement. If you have a number of values like for example route prefixes, it is more efficient to refer to a policy list instead of creating one match rule per prefix.

Policy lists are configured separately and can thereby be maintained more easily. Besides, policy lists can be referenced by multiple policy statements.

1.2.3. Conditions

Policy conditions are configured separately, and can be used as an additional option in policy statements. A policy rule (ordinal) will only be executed if the condition is true. Conditional policies allow to make policy execution depended on certain states of the system, for example:

- Protocol neighbor states
- Presence or absence of a specific route and/or path attribute
- Number of routes in a routing table

One condition is supported per ordinal. A single condition can be attached to multiple policies.

1.2.4. Attachment Points

Policies define set of rules that can be used for various purposes by various applications. Once policies have been created, they need to be applied in order to take effect. Attachment points describe the specific applications and processes to which policies can be applied. RBFS currently supports the following policy attachment points:

- Instance import/export - Policies attached to an instance at the address family level allow to control which routes will be imported into the instance and exported from the instance by BGP. Such import and export policies are commonly used with BGP L3VPNs.
- BGP peer group import/export - Policies attached to BGP peer groups allow to define which routes will be advertised to and accepted from BGP peers. You can attach policies to both instances or peer groups to define the import and export behaviour. You can also combine both attachment points, for example if some policy rules apply generically to the instance and some other rules specifically to a peer or peer group.
- BGP redistribution - You can attach policies to BGP redistribution to define which routes will be redistributed into the BGP process. This is useful if you would like to redistribute only a sub-set of a type of routes.
- IS-IS redistribution - Policies can be attached to IS-IS redistribution to control which routes will be redistributed into the IS-IS process.
- IGMP group filtering - You can apply policies to IGMP interface profiles. Such policies act as IGMP group filters when receiving IGMP Membership Report messages.
- IGMP SSM-mapping - Policies are further used to define SSM mapping. SSM mapping policies attached to IGMP interface profiles define how to translate IGMPv2 (*,G) to IGMPv3 (S,G) Reports.

1.3. Policy Behaviour

The default behaviour of a policy is deny. This means, if a route or any other object is subject to a policy, by default it will be marked as deny. For example in case of an import policy, it will not be imported. A policy will permit an object, for example import a route, if the route or object successfully matches the match rules, and if there is an action rule with a permit. In addition, further operations like modifying

route attributes might be executed.

Policy ordinals are executed in the order of the ordinal numbers.

- If an ordinal results in a terminating action like permit or deny, the policy processing is completed for the respective object. Subsequent ordinals will not be processed.
- If an ordinal does not result in a match, the next ordinal is processed.

There might be situations in which a policy configuration is not complete or not valid. In particular, a policy may contain a match or action type not supported by an attachment point. For example, the ipv4-mcast-group type is not supported by BGP or IS-IS. The following list summarizes the behaviour for such invalid scenarios:

- If a policy is attached but does not exist, all routes or objects will be denied.
- If a policy contains only statements, or only statements and ordinals, but no match and action block, it will deny all.
- If a match or action type is not supported by the attachment point, the policy will ignore the unsupported rule and process only the supported rules. For any ignored rule, the default deny is not impacted. The behaviour will be the same as if the unsupported rule does not exist.

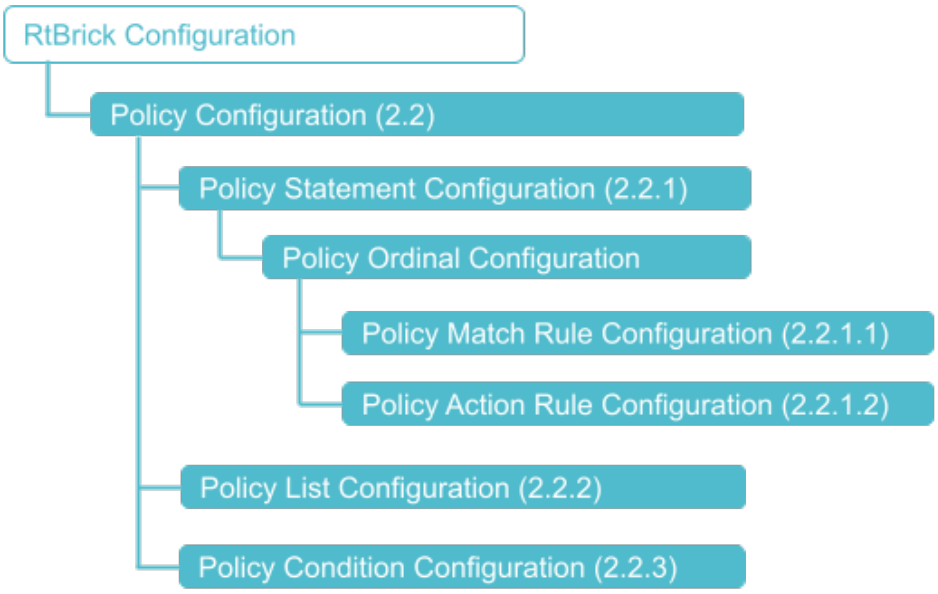
1.4. Supported Platforms

Not all features are necessarily supported on each hardware platform. Please refer to the RBFS Platform Guide for the features and the sub-features supported or not supported for each platform.

2. Policy Configuration

2.1. Configuration Hierarchy

The diagram illustrates the policy configuration hierarchy.



2.2. Configuration Syntax and Commands

The following sections describe the policy configuration syntax and commands.

2.2.1. Configuring Policy Statements

Syntax:

set policy statement <policy-name> **ordinal** <number> <attribute> <value>

Attribute	Description
<policy-name>	Name of the policy statement. Policy names can contain alphanumeric characters and underscore character. They must not include special characters like hyphen. For example, **BGP-EXPORT** is not supported, whereas **BGP_EXPORT** is supported. A valid name cannot start with a number but it can contain numbers and underscore (_) in the string. The length of the names should not exceed 64 characters.
<number>	Specifies the ordinal number.
description <text>	Description of the ordinal.

Attribute	Description
match <...>	Match configuration hierarchy. Please refer to section 2.2.1.1 for the match rule configuration.
match-logic <value>	Specifies the match logic. Supported values are and and or.
action <...>	Action configuration hierarchy. Please refer to section 2.2.1.2 for the action rule configuration.
condition <condition-name>	Optionally, apply a policy condition. Please refer to section 2.2.3 for the policy condition configuration.

2.2.1.1. Configuring Match Rules

Syntax:

set policy statement <policy-name> **ordinal** <number> **match rule** <number> <attribute> <value>

Attribute	Description
rule <number>	Specifies the match rule number.
type <attribute-type>	Specifies the attribute type. Please refer to section 2.2.1.1.1 for supported attribute types.
match-type <match-type>	Specifies the match type. Please refer to section 2.2.1.1.1 for supported match types per attribute, and to section 2.2.1.1.2 for descriptions of the match types.
value <value>	Attribute value. This is the actual value of the attribute to match, for example an IP prefix, a metric, or a community.
value-type <value-type>	Attribute value type. Supported types are discrete, this is a single value, and list, a list of values defined in a policy list.

2.2.1.1.1. Attribute and Match Types

Attribute Type	Match Types Supported
ipv4-prefix	regex exact longer or-longer prefix-length-exact prefix-length-greater prefix-length-greater-or-exact

Attribute Type	Match Types Supported
ipv6-prefix	regex exact longer or-longer prefix-length-exact prefix-length-greater prefix-length-greater-or-exact
route-distinguisher	regex exact
community	regex exact exists
extended-community	regex exact exists
large-community	regex exact exists
as-path	regex exact exists
cluster-list	regex exact exists
source	regex exact
sub-source	regex exact
originator-identifier	regex exact
peer-router-id	regex exact
ipv4-nexthop	regex exact
ipv6-nexthop	regex exact
label	regex exact
peer-ipv4	regex exact

Attribute Type	Match Types Supported
peer-ipv6	regex exact
sid	regex exact
sid-flag	regex exact
external	exact

2.2.1.1.2. Match Types

Match Types	Description
regex	An attribute can be matched using a standard Linux egrep regular expression. Example: "label": "label-op:push,label:206,bos:1" In this example, the label is a 64bit number, which has label value, bos, and operation encoding. A regex is used to match the string which is displayed in the table dump, that is, label-op:push,label:206,bos:1 not the 64bit value. The same is applicable to an array type attribute. A regex can be written to the string which is visible in the table dump output.
exact	Value configured in the command must be same as application attribute value
exists	This is applicable only for array type attribute; an exist match is the one where value configured in the command must exist in the application attribute value which is an array.
less	The application attribute value must be less than the value configured in the command
less-or-exact	The application attribute value must be less than or exact value configured in the command
greater	The application attribute value must be greater than the value configured in the command
greater-or-exact	The application attribute value must be greater than or exact value configured in the command
greater-longer	The route shares the same most-significant bits (described by prefix-length), and prefix-length is greater than the route's prefix length

Match Types	Description
greater-or-longer	The route shares the same most-significant bits (described by prefix-length), and prefix-length is equal to or greater than the route's prefix length.
longer	The route address shares the same most-significant bits as the match prefix (destination-prefix or source-prefix). The number of significant bits is described by the prefix-length component of the match prefix.
or-longer	The route address shares the same most-significant bits as the match prefix (destination-prefix or the source-prefix). The number of significant bits is described by the prefix-length component of the match prefix.
prefix-length-exact	The application attribute value whose prefix length must be lesser than or exact the value configured in the command
prefix-length-greater	The application attribute value whose prefix length must be greater than the value configured in the command
prefix-length-greater-or-exact	The application attribute value whose prefix length must be greater than or exact value configured in the command

2.2.1.2. Configuring Action Rules

Syntax:

set policy statement <policy-name> **ordinal** <number> **action rule** <number> <attribute> <value>

Attribute	Description
rule <number>	Specifies the action rule number.
operation <operation-type>	Specifies the operation type. Please refer to section 2.2.1.2.1 for supported operations, and to section 2.2.1.2.2 for operations per attribute.
type <attribute-type>	Specifies the attribute type. Please refer to section 2.2.1.2.2 for supported attribute types.
value <value>	Specifies the operation value.

2.2.1.2.1. Operation Types

Operation Type	Description
add	The application attribute value will be added with the value configured in the command

Operation Type	Description
append	The application attribute value will be appended with the value configured in the command
delete-attribute	Deletes the attribute from the route/BDS object, that is, clearing all the info for that specific attribute in the object
divide	The application attribute value will be divided with the value configured in the command
goto-next-ordinal	If next term exists, then next term is executed and the policy result is decided based on the result of the execution
multiply	The application attribute value will be multiplied with the value configured in the command
overwrite	The application attribute value will be overwritten with the value configured in the command
prepend	The application attribute value will be prefixed with the value configured in the command
return-deny	Stops policy execution and returns result as deny (resulting route/BDS object to be denied)
return-permit	Stops policy execution and return result as permit (resulting route/BDS object to be permitted)
subtract	The application attribute value will be subtracted with the value configured in the command. If the result of the subtraction results in a number less than 0, the value "0" is used.

2.2.1.2.2. Attribute Types and Supported Operations

Attribute Type	Operation Types Supported
ipv4-prefix	overwrite
ipv6-prefix	overwrite
route-distinguisher	overwrite
community	append prepend overwrite
extended-community	append prepend overwrite
large-community	append prepend overwrite

Attribute Type	Operation Types Supported
as-path	append prepend overwrite
cluster-list	append prepend overwrite
source	overwrite
sub-source	overwrite
originator-identifier	overwrite
peer-router-id	overwrite
ipv4-nexthop	overwrite
ipv6-nexthop	overwrite
label	overwrite
peer-ipv4	overwrite
peer-ipv6	overwrite
sid	overwrite
sid-flag	overwrite
external	overwrite

Example: Policy statement configuration

```
{
    "rtbrick-config:policy": {
      "statement": [
        {
          "name": "EXPORT_POLICY1",
          "ordinal": [
            {
              "ordinal": 10,
              "description": "Add BGP community to direct routes",
              "match-logic": "and",
              "match": {
                "rule": [
                  {
                    "rule": 1,
                    "type": "ipv6-prefix",
                    "value-type": "discrete",
                    "match-type": "or-longer",
                    "value": "fd3d:3d:0:99::/64"
                  },
                  {
                    "rule": 2,
                    "type": "source",
                    "value-type": "discrete",
```

```
                    "match-type": "exact",
                    "value": "direct"
                  }
                ]
              },
              "action": {
                "rule": [
                  {
                    "rule": 1,
                    "type": "community",
                    "operation": "append",
                    "value": "100:1"
                  },
                  {
                    "rule": 2,
                    "operation": "return-permit"
                  }
                ]
              }
            },
            {
              "ordinal": 20,
              "description": "Allow any other route",
              "action": {
                "rule": [
                  {
                    "rule": 1,
                    "operation": "return-permit"
                  }
                ]
              }
            }
          ]
        }
      ]
    }
  }
```

2.2.2. Configuring Policy Lists

Syntax:

set policy list <list-name> <list-type> **ordinal** <ordinal-number> **value** <value>

Attribute	Description
<list-name>	Name of the policy list

Attribute	Description
<list-type>	Type of the policy list. The following types of lists are supported: * as-path * cluster-list * community * route-distinguisher * extended-community * ipv4-address * ipv4-mcast-group * ipv4-prefix * ipv6-address * ipv6-prefix * large-community * mac-address * mpls-label * source * sub-source
<ordinal-number>	The number of the list entry.
<value>	The value of the list entry, for example an IP prefix, or a community.

Example: Policy list configuration

```
{
    "rtbrick-config:policy": {
      "list": [
        {
          "name": "PREFIX_LIST1",
          "type": "ipv6-prefix",
          "ordinal": [
            {
              "ordinal": 1,
              "value": "fd3d:3d:1000::/56"
            },
            {
              "ordinal": 2,
              "value": "fd3d:3d:a:10::/64"
            },
            {
              "ordinal": 3,
              "value": "fd3d:3d:d:30::/64"
            }
          ]
        }
      ]
    }
}
```

2.2.3. Configuring Policy Conditions

Policy conditions refer to certain states of the system represented in BDS tables. You need to specify the table, the daemon (BD) that needs to resolve the condition, and if applicable, the attributes used to define the condition.

There are two types of conditions:

- Match on certain attributes in BDS table objects.
- Match on the number of objects in a BDS table.

2.2.3.1. Configuring Table Match

Syntax:

set policy condition <condition-name> **table** <attribute> <value>

Attribute	Description
<condition-name>	Name of the policy condition
name <table-name>	Name of the BDS table

Attribute	Description
bd <bd-name>	Name of the BD which resolves the condition. Currently supported BDs are: ifmd, ribd, mribd, pppoed, subscriberd, ipoed, l2tpd, pimd, igmp.iod, isis.iod, ospf.appd ospf.iod
count <number>	Optionally, match on the number of objects in a table
match <type>	Type of match for an object count. Supported match types are: equal greater greater-or-equal less less-or-equal

2.2.3.2. Configuring Attribute Match

You can configure attributes to define a condition. The attributes refer to objects in the table configured in the section above. Please note:

- You can match on multiple attributes.
- In order to identify matching objects, you need to specify all attributes which are primary keys in the table.
- Attribute configuration is not required for conditions matching on the number of table objects using the count option.

Syntax:

set policy condition <condition-name> **attribute** <name> <attribute> <value>

Attribute	Description
<condition-name>	Name of the policy condition
attribute <name>	Name of the attribute in the BDS table
value <value>	Value of the attribute to match
match <type>	Type of the match. Supported match types are: equal exists greater greater-or-equal less less-or-equal regex

Example: Policy condition configuration

```
{
    "rtbrick-config:policy": {
      "condition": [
        {
          "condition_name": "precheck",
          "table": {
            "name": "global.instance",
            "bd": "bgp.iod.1"
          },
          "attribute": [
            {
              "name": "instance_name",
              "match": "equal",
              "value": "default"
            }
          ]
        }
      ]
    }
  }
```

2.2.4. Attaching Policies

Once a policy has been created, they need to be applied to an application like a routing protocol to take effect.

2.2.4.1. BGP Attachment Points

- Instance import/export
- BGP peer group import/export
- BGP redistribution

For attaching policies to the BGP protocol, please refer to the RBFS BGP User Guide.

2.2.4.2. IS-IS Attachment Points

- IS-IS redistribution

For attaching policies to the IS-IS protocol, please refer to the RBFS IS-IS User Guide.

2.2.4.3. IGMP Attachment Points

- IGMP group filtering
- IGMP SSM-mapping

For attaching policies to the IGMP protocol, please refer to the RBFS IP Multicast Routing Configuration Guide.

2.3. Sample Configurations

Example 1: BGP export policy referencing a policy list

```
supervisor@leaf1: cfg> show config policy
{
  "rtbrick-config:policy": {
    "list": [
      {
        "name": "PREFIX_LIST2",
        "type": "ipv6-prefix",
        "ordinal": [
          {
            "ordinal": 1,
            "value": "fd3d:3d:0:99::/64"
          },
          {
            "ordinal": 2,
            "value": "fd3d:3d:100:a::/64"
          },
          {
            "ordinal": 3,
            "value": "fd3d:3d:c0:a8::/64 "
          }
        ]
      }
    ],
    "statement": [
      {
        "name": "EXPORT_POLICY2",
        "ordinal": [
          {
            "ordinal": 10,
            "description": "Add community to direct routes",
            "match": {
              "rule": [
                {
                  "rule": 1,
                  "type": "source",
                  "value-type": "discrete",
                  "match-type": "exact",
                  "value": "direct"
                }
              ]
            },
            "action": {
              "rule": [
                {
                  "rule": 1,
                  "type": "community",
                  "operation": "append",
                  "value": "100:1"
```

```
                },
                {
                  "rule": 2,
                  "operation": "return-permit"
                }
              ]
            }
          },
          {
            "ordinal": 20,
            "description": "Allow list of routes",
            "match": {
              "rule": [
                {
                  "rule": 1,
                  "type": "ipv6-prefix",
                  "value-type": "list",
                  "match-type": "or-longer",
                  "value": "PREFIX_LIST2"
                }
              ]
            },
            "action": {
              "rule": [
                {
                  "rule": 1,
                  "operation": "return-permit"
                }
              ]
            }
          },
          {
            "ordinal": 30,
            "description": "Deny any other route",
            "action": {
              "rule": [
                {
                  "rule": 1,
                  "operation": "return-deny"
                }
              ]
            }
          }
        ]
      }
    ]
  }
}

supervisor@leaf1: cfg> show config instance default protocol bgp peer-group
spine address-family ipv6 unicast
{
  "rtbrick-config:address-family": [
    {
      "afi": "ipv6",
      "safi": "unicast",
      "policy": {
        "export": "EXPORT_POLICY2"
```

```
        }
      }
    ]
}
```

Example 2: IGMP filter policy

```
supervisor@leaf1: cfg> show config policy
{
  "rtbrick-config:policy": {
    "statement": [
      {
        "name": "IGMP_FILTER",
        "ordinal": [
          {
            "ordinal": 1,
            "description": "IGMP group filter",
            "match-logic": "or",
            "match": {
              "rule": [
                {
                  "rule": 1,
                  "type": "ipv4-mcast-group",
                  "value-type": "discrete",
                  "match-type": "or-longer",
                  "value": "232.100.0.0/24"
                },
                {
                  "rule": 2,
                  "type": "ipv4-mcast-group",
                  "value-type": "discrete",
                  "match-type": "or-longer",
                  "value": "232.200.0.0/24"
                }
              ]
            },
            "action": {
              "rule": [
                {
                  "rule": 1,
                  "operation": "return-permit"
                }
              ]
            }
          }
        ]
      }
    ]
  }
}

supervisor@leaf1: cfg> show config multicast-options igmp
{
  "rtbrick-config:igmp": {
    "interface-profile": [
      {
        "profile-name": "PROFILE1",
        "filter-policy": "IGMP_FILTER"
      }
    ]
  }
}
```

3. Operational Commands

3.1. Policy Test

You can use the policy test feature to test a policy before attaching it to a protocol or an instance.

Perform the following tasks:

- Step 1: Identify the brick daemon that will process the policy and the table to which the policy will be applied.
- Step 2: Execute the 'test policy run' command.

Example: Testing a BGP VPN export policy

```
supervisor@leaf1: op> test policy run bgp.appd.1 policy-name VPN_V4_EXPORT
table default.bgp.rib-in.import.ipv4.vpn-unicast
```

- Step 3: View the test results.

The policy test feature will create two result tables. The result table ending with ".policy.permit" will show all objects permitted by the policy, the one ending with ".policy.deny" will show all objects denied by the policy.

Example: Viewing the result tables

```
supervisor@leaf1: op> show datastore bgp.appd.1 table default.bgp.rib-
in.import.ipv4.vpn-unicast.policy.permit
<...>
supervisor@leaf1: op> show datastore bgp.appd.1 table default.bgp.rib-
in.import.ipv4.vpn-unicast.policy.deny
<...>
```

- Step 4: Clear the result tables

You can clear the result tables using the 'test policy clear' command. Apply the clear command to the same table for which you have run the policy test.

Example: Clearing the result tables

```
supervisor@leaf1: op> test policy clear bgp.appd.1 policy-name VPN_V4_EXPORT
table default.bgp.rib-in.import.ipv4.vpn-unicast
```