

Chapter 8

Configuration Groups

This chapter discusses the following topics:

Overview on page 228

Configuration Groups Configuration Statements on page 228

Configuration Groups Configuration Guidelines on page 229

Examples: Configuration Groups on page 238

Summary of Configuration Group Statements on page 249

Overview

Configuration groups allow you to create a group containing configuration statements and to direct the inheritance of that group's statements in the rest of the configuration. The same group can be applied to different sections of the configuration, and different sections of one group's configuration statements can be inherited in different places in the configuration.

Configuration groups allow you to create smaller, more logically constructed configuration files, making it easier to configure and maintain the JUNOS software. For example, you can group statements that are repeated in many places in the configuration, such as when configuring interfaces, and thereby limit updates to just the group.

You can also use wildcards in a configuration group to allow configuration data to be inherited by any object that matches a wildcard expression.

The configuration group mechanism is separate from the grouping mechanisms used elsewhere in the configuration, such as BGP groups. Configuration groups provide a generic mechanism that can be used throughout the configuration but that are known only to the JUNOS software command-line interface (CLI). The individual software processes that perform the actions directed by the configuration receive the expanded form of the configuration; they have no knowledge of configuration groups.

Inheritance Model

Configuration groups use true inheritance, which involves a dynamic, ongoing relationship between the source of the configuration data and the target of that data. Data values changed in the configuration group are automatically inherited by the target. The target need not contain the inherited information, although the inherited values can be overridden in the target without affecting the source from which they were inherited.

This inheritance model allows you to see only the instance-specific information without seeing the inherited details. A command pipe in configuration mode allows you to display the inherited data.

Configuration Groups Configuration Statements

To configure configuration groups and inheritance, you can include the following statements in the configuration:

```
groups {
  group-name {
    configuration-data;
  }
}
```

Include the `apply-groups [group-names]` statement anywhere in the configuration that the configuration statements contained in a configuration group are needed.

Configuration Groups Configuration Guidelines

For areas of your configuration to inherit configuration statements, you must first put the statements into a configuration group and then apply that group to the levels in the configuration hierarchy that require the statements. This section covers the following topics:

Create a Configuration Group on page 229

Apply a Configuration Group on page 230

Disable Inheritance of a Configuration Group on page 232

Display Inherited Values on page 233

Use Wildcards on page 234

Create a Configuration Group

To create a configuration group, include the groups statement at the [edit] hierarchy level:

```
groups {
  group-name {
    configuration-data;
  }
}
```

group-name is the name of a configuration group. To configure multiple groups, specify more than one *group-name*. On routers that support multiple Routing Engines, you can also specify two special group names:

re0—Configuration statements applied to the Routing Engine in slot 0.

re1—Configuration statements applied to the Routing Engine in slot 1.

The configuration specified in group re0 is only applied if the current Routing Engine is in slot 0; likewise, the configuration specified in group re1 is only applied if the current Routing Engine is in slot 1. Therefore, both Routing Engines can use the same configuration file, each using only the configuration statements that apply to it. Each re0 or re1 group contains at a minimum the configuration for the hostname and the management interface (fxp0). If each Routing Engine uses a different management interface, the group also should contain the configuration for the backup router and static routes.

configuration-data contains the configuration statements applied elsewhere in the configuration with the apply-groups statement. To have a configuration inherit the statements in a configuration group, include the apply-groups statement. For information about the apply-groups statement, see the “Apply a Configuration Group” on page 230.

Apply a Configuration Group

To have a configuration inherit the statements in a configuration group, include the `apply-groups` statement:

```
apply-groups [ group-name ];
```

If you specify more than one group name, list them in order of inheritance priority. The configuration data in the first group takes priority over the data in subsequent groups.

For routers that support multiple Routing Engines, you can specify `re0` and `re1` as group names. The configuration specified in group `re0` is only applied if the current Routing Engine is in slot 0; likewise, the configuration specified in group `re1` is only applied if the current Routing Engine is in slot 1. Therefore, both Routing Engines can use the same configuration file, each using only the configuration statements that apply to it. Each `re0` or `re1` group contains at a minimum the configuration for the hostname and the management interface (`fxp0`). If each Routing Engine uses a different management interface, the group also should contain the configuration for the backup router and static routes.

You can include the `apply-groups` statement at any level of the configuration hierarchy, listing group names within each `apply-groups` statement in priority order.

You can include only one `apply-groups` statement at each specific level of the configuration hierarchy. The `apply-groups` statement at a specific hierarchy level lists the configuration groups to be added to the containing statement's list of configuration groups.

Values specified at the specific hierarchy level override values inherited from the configuration group.

Groups listed in nested `apply-groups` statements take priority over groups in outer statements. In the following example, the BGP `neighbor 10.0.0.1` inherits configuration data from group one first, then from groups two and three. Configuration data in group one overrides data in any other group. Data from group ten is used only if a statement is not contained in any other group.

```
apply-groups [ eight nine ten ];
protocols {
  apply-groups seven;
  bgp {
    apply-groups [ five six ];
    group some-bgp-group {
      apply-groups four;
      neighbor 10.0.0.1 {
        apply-groups [ one two three ];
      }
    }
  }
}
```

Example: Configure and Apply Configuration Groups

In this example, the SNMP configuration is divided between the group basic and the normal configuration hierarchy.

There are a number of advantages to placing the system-specific configuration (SNMP contact) into a configuration group and thus separating it from the normal configuration hierarchy—the user can replace (using the load replace command) either section without discarding data from the other.

In addition, setting a contact for a specific box is now possible because the group data would be hidden by the router-specific data.

```
[edit]
groups {
    # "groups" is a top-level statement
    basic {
        # User defined group name
        snmp {
            # This group contains some snmp data
            contact "My Engineering Group";
            community BasicAccess {
                authorization read-only;
            }
        }
    }
}
apply-groups basic; # Enable inheritance from group "basic"
snmp {
    # Some normal (non-group) configuration
    location "West of Nowhere";
}
```

This configuration is equivalent to the following:

```
[edit]
snmp {
    location "West of Nowhere";
    contact "My Engineering Group";
    community BasicAccess {
        authorization read-only;
    }
}
```

For information about how to disable inheritance of a configuration group, see “Disable Inheritance of a Configuration Group” on page 232.

Disable Inheritance of a Configuration Group

To disable inheritance of a configuration group at any level except the top level of the hierarchy, include the `apply-groups-except` statement:

```
apply-groups-except [ group-name ];
```

This is useful when you use the `apply-group` statement at a specific hierarchy level but also want to override the values inherited from the configuration group for a specific parameter.

Example: Disable Inheritance on Interface s0-1/1/0

In the following example, the `apply-groups` statement is applied globally at the interfaces level. The `apply-groups-except` statement is also applied at interface `s0-1/1/0` so that it uses the default values `hold-time` and `link-mode`.

```
[edit]
groups {
  global {
    interfaces {
      <*> {
        hold-time down 640;
        link-mode full-duplex;
      }
    }
  }
}
apply-groups global;
interfaces {
  so-1/1/0 {
    apply-groups-except global; # Disables inheritance from group "global"
  }
  # Interface uses default value "hold-time" and
  "link-mode"
}
```

For information about applying a configuration group, see “Apply a Configuration Group” on page 230.

Display Inherited Values

Configuration groups can add some confusion regarding the actual values used by the router, because configuration data can be inherited from configuration groups. To view the actual values used by the router, use the `display inheritance` command after the pipe in a `show` command. This command displays the inherited statements at the level at which they are inherited and the group from which they have been inherited.

```
[edit]
user@host# show | display inheritance
snmp {
  location "West of Nowhere";
  ##
  ## 'My Engineering Group' was inherited from group 'basic'
  ##
  contact "My Engineering Group";
  ##
  ## 'BasicAccess' was inherited from group 'basic'
  ##
  community BasicAccess {
    ##
    ## 'read-only' was inherited from group 'basic'
    ##
    authorization read-only;
  }
}
```

To display the expanded configuration (the configuration, including the inherited statements) without the `##` lines, use the `except ##` command after the pipe in a `show` command:

```
[edit]
user@host# show | display inheritance | except ##
snmp {
  location "West of Nowhere";
  contact "My Engineering Group";
  community BasicAccess {
    authorization read-only;
  }
}
```

Use Wildcards

You can use wildcards to identify names and allow one statement to provide data for a variety of statements. For example, grouping the configuration of the sonet-options statement over all SONET/SDH interfaces or the dead interval for OSPF over all ATM interfaces simplifies configuration files and eases their maintenance.

Wildcarding in normal configuration data is done in a style that is consistent with traditional UNIX shell name wildcarding. In this style of wildcarding, you can use the following metacharacters:

Asterisk (*)—Matches any string of characters.

Question mark (?)—Matches any single character.

Open bracket ([)—Introduces a character class.

Close bracket (])—Indicates the end of a character class. If the close bracket is missing, the open bracket matches a [rather than introduce a character class.

A character class matches any of the characters between the square brackets. Character classes must be enclosed in quotation marks (" ").

Hyphen (-)—Specifies a range of characters.

Exclamation point (!)—The character class can be complemented by making an exclamation point the first character of the character class. To include a] in a character class, make it the first character listed (after the !, if any). To include a minus sign, make it the first or last character listed.

Wildcarding in configuration groups follows the same rules, but the wildcard pattern must be enclosed in angle brackets (<pattern>) to differentiate it from other wildcarding in the configuration file. For example:

```
[edit]
groups {
  sonet-default {
    interfaces {
      <so-*> {
        sonet-options {
          payload-scrambler;
          rfc-2615;
        }
      }
    }
  }
}
```

Wildcard expressions match (and provide configuration data for) existing statements in the configuration that match their expression only. In the example above, the expression <so-*> passes its sonet-options statement to any interface that matches the expression so-*.

Angle brackets allow you to pass normal wildcarding through without modification. In all matching within the configuration, whether it is done with or without wildcards, the first item encountered in the configuration that matches is used. In the following example, data from the wildcarded BGP groups is inherited in the order in which the groups are listed. The preference value from <*a*> overrides the preference in <*b*>, just as the p value from <*c*> overrides the one from <*d*>. Data values from any of these groups override the data values from abcd.

```
[edit]
user@host# show
groups {
  one {
    protocols {
      bgp {
        group <*a*> {
          preference 1;
        }
        group <*b*> {
          preference 2;
        }
        group <*c*> {
          out-delay 3;
        }
        group <*d*> {
          out-delay 4;
        }
        group abcd {
          preference 10;
          hold-time 10;
          out-delay 10;
        }
      }
    }
  }
}
protocols {
  bgp {
    group abcd {
      apply-groups one;
    }
  }
}
```

```
[edit]
user@host# show | display inheritance
protocols {
  bgp {
    group abcd {
      ##
      ## '1' was inherited from group 'one'
      ##
      preference 1;
      ##
      ## '10' was inherited from group 'one'
      ##
      hold-time 10;
      ##
      ## '3' was inherited from group 'one'
      ##
      out-delay 3;
    }
  }
}
```

Example: Use Wildcards

The following example demonstrates the use of wildcarding. The interface so-0/0/0 inherits data from the various SONET/SDH interface wildcard patterns in group one.

```
[edit]
user@host# show
groups {
  one {
    interfaces {
      <so-*> {
        sonet-options {
          rfc-2615;
        }
      }
      <so-0/*> {
        sonet-options {
          fcs 32;
        }
      }
      <so-*/0/*> {
        sonet-options {
          fcs 16;
        }
      }
      <so-*/*/0> {
        sonet-options {
          payload-scrambler;
        }
      }
    }
  }
}
```

```

apply-groups one;
interfaces {
  so-0/0/0 {
    unit 0 {
      family inet {
        address 10.0.0.1/8;
      }
    }
  }
}
[edit]
user@host# show | display inheritance
interfaces {
  so-0/0/0 {
    ##
    ## 'sonet-options' was inherited from group 'one'
    ##
    sonet-options {
      ##
      ## '32' was inherited from group 'one'
      ##
      fcs 32;
      ##
      ## 'payload-scrambler' was inherited from group 'one'
      ##
      payload-scrambler;
      ##
      ## 'rfc-2615' was inherited from group 'one'
      ##
      rfc-2615;
    }
    unit 0 {
      family inet {
        address 10.0.0.1/8;
      }
    }
  }
}

```

Examples: Configuration Groups

The following examples illustrate ways to use configuration groups and inheritance:

Configure Sets of Statements on page 238

Configure Interfaces on page 240

Configure Peer Entities on page 243

Establish Regional Configurations on page 245

Select Wildcard Names on page 247

Configure Sets of Statements

When sets of statements exist in configuration groups, all values are inherited. For example:

```
[edit]
user@host# show
groups {
  basic {
    snmp {
      interface so-1/1/1.0;
    }
  }
}
apply-groups basic;
snmp {
  interface so-0/0/0.0;
}
[edit]
user@host# show | display inheritance
snmp {
  ##
  ## 'so-1/1/1.0' was inherited from group 'basic'
  ##
  interface [ so-0/0/0.0 so-1/1/1.0 ];
}
```

For sets that are not displayed within brackets, all values are also inherited. For example:

```
[edit]
user@host# show
groups {
  worldwide {
    system {
      name-server {
        10.0.0.100;
        10.0.0.200;
      }
    }
  }
}
```

```
apply-groups worldwide;
system {
  name-server {
    10.0.0.1;
    10.0.0.2;
  }
}
```

[edit]

user@host# **show | display inheritance**

```
system {
  name-server {
    10.0.0.1;
    10.0.0.2;
    ##
    ## '10.0.0.100' was inherited from group 'worldwide'
    ##
    10.0.0.100;
    ##
    ## '10.0.0.200' was inherited from group 'worldwide'
    ##
    10.0.0.200;
  }
}
```

Configure Interfaces

You can use configuration groups to separate the common interface media parameters from the interface-specific addressing information. The following example places configuration data for ATM interfaces into a group called atm-options:

```
[edit]
user@host# show
groups {
  atm-options {
    interfaces {
      <at-*> {
        atm-options {
          vpi 0 maximum-vcs 1024;
        }
        unit <*> {
          encapsulation atm-snap;
          point-to-point;
          family iso;
        }
      }
    }
  }
}
apply-groups atm-options;
interfaces {
  at-0/0/0 {
    unit 100 {
      vci 0.100;
      family inet {
        address 10.0.0.100/30;
      }
    }
    unit 200 {
      vci 0.200;
      family inet {
        address 10.0.0.200/30;
      }
    }
  }
}
```

```

[edit]
user@host# show | display inheritance
interfaces {
  at-0/0/0 {
    ##
    ## "atm-options" was inherited from group "atm-options"
    ##
    atm-options {
      ##
      ## "1024" was inherited from group "atm-options"
      ##
      vpi 0 maximum-vcs 1024;
    }
    unit 100 {
      ##
      ## "atm-snap" was inherited from group "atm-options"
      ##
      encapsulation atm-snap;
      ##
      ## "point-to-point" was inherited from group "atm-options"
      ##
      point-to-point;
      vci 0.100;
      family inet {
        address 10.0.0.100/30;
      }
      ##
      ## "iso" was inherited from group "atm-options"
      ##
      family iso;
    }
    unit 200 {
      ##
      ## "atm-snap" was inherited from group "atm-options"
      ##
      encapsulation atm-snap;
      ##
      ## "point-to-point" was inherited from group "atm-options"
      ##
      point-to-point;
      vci 0.200;
      family inet {
        address 10.0.0.200/30;
      }
      ##
      ## "iso" was inherited from group "atm-options"
      ##
      family iso;
    }
  }
}

```

```
[edit]
user@host# show | display inheritance | except ##
interfaces {
  at-0/0/0 {
    atm-options {
      vpi 0 maximum-vc 1024;
    }
    unit 100 {
      encapsulation atm-snap;
      point-to-point;
      vci 0.100;
      family inet {
        address 10.0.0.100/30;
      }
      family iso;
    }
    unit 200 {
      encapsulation atm-snap;
      point-to-point;
      vci 0.200;
      family inet {
        address 10.0.0.200/30;
      }
      family iso;
    }
  }
}
```

Configure Peer Entities

In this example, we create a group `some-isp` that contains configuration data relating to another ISP. We can then insert `apply-group` statements at any point to allow any location in the configuration hierarchy to inherit this data.

```
[edit]
user@host# show
groups {
  some-isp {
    interfaces {
      <ge-*> {
        gigheter-options {
          flow-control;
        }
      }
    }
  }
  protocols {
    bgp {
      group <*> {
        neighbor <*> {
          remove-private;
        }
      }
    }
    pim {
      interface <*> {
        version 1;
      }
    }
  }
}
interfaces {
  ge-0/0/0 {
    apply-groups some-isp;
    unit 0 {
      family inet {
        address 10.0.0.1/24;
      }
    }
  }
}
protocols {
  bgp {
    group main {
      neighbor 10.254.0.1 {
        apply-groups some-isp;
      }
    }
  }
  pim {
    interface ge-0/0/0.0 {
      apply-groups some-isp;
    }
  }
}
```

```

[edit]
user@host# show | display inheritance
interfaces {
  ge-0/0/0 {
    ##
    ## "gigether-options" was inherited from group "some-isp"
    ##
    gigether-options {
      ##
      ## "flow-control" was inherited from group "some-isp"
      ##
      flow-control;
    }
    unit 0 {
      family inet {
        address 10.0.0.1/24;
      }
    }
  }
}
protocols {
  bgp {
    group main {
      neighbor 10.254.0.1 {
        ##
        ## "remove-private" was inherited from group "some-isp"
        ##
        remove-private;
      }
    }
  }
}
pim {
  interface ge-0/0/0.0 {
    ##
    ## "1" was inherited from group "some-isp"
    ##
    version 1;
  }
}
}

```

Establish Regional Configurations

In this example, one group is populated with configuration data that is standard throughout the company, while another group contains regional deviations from this standard:

```
[edit]
user@host# show
groups {
  standard {
    interfaces {
      <t3-*> {
        t3-options {
          compatibility-mode larscom subrate 10;
          idle-cycle-flag ones;
        }
      }
    }
  }
  northwest {
    interfaces {
      <t3-*> {
        t3-options {
          long-buildout;
          compatibility-mode kentrox;
        }
      }
    }
  }
}
apply-groups standard;
interfaces {
  t3-0/0/0 {
    apply-groups northwest;
  }
}
```

```
[edit]
user@host# show | display inheritance
interfaces {
  t3-0/0/0 {
    ##
    ## "t3-options" was inherited from group "northwest"
    ##
    t3-options {
      ##
      ## "long-buildout" was inherited from group "northwest"
      ##
      long-buildout;
      ##
      ## "kentrox" was inherited from group "northwest"
      ##
      compatibility-mode kentrox;
      ##
      ## "ones" was inherited from group "standard"
      ##
      idle-cycle-flag ones;
    }
  }
}
```

Select Wildcard Names

You can combine wildcarding and thoughtful use of names in statements to tailor statement values:

```
[edit]
user@host# show
groups {
  mpls-conf {
    protocols {
      mpls {
        label-switched-path <*-major> {
          retry-timer 5;
          bandwidth 155m;
          optimize-timer 60;
        }
        label-switched-path <*-minor> {
          retry-timer 15;
          bandwidth 64k;
          optimize-timer 120;
        }
      }
    }
  }
}
apply-groups mpls-conf;
protocols {
  mpls {
    label-switched-path metro-major {
      to 10.0.0.10;
    }
    label-switched-path remote-minor {
      to 10.0.0.20;
    }
  }
}
```

```
[edit]
user@host# show | display inheritance
protocols {
  mpls {
    label-switched-path metro-major {
      to 10.0.0.10;
      ##
      ## "5" was inherited from group "mpls-conf"
      ##
      retry-timer 5;
      #
      ## "155m" was inherited from group "mpls-conf"
      ##
      bandwidth 155m;
      ##
      ## "60" was inherited from group "mpls-conf"
      ##
      optimize-timer 60;
    }
    label-switched-path remote-minor {
      to 10.0.0.20;
      ##
      ## "15" was inherited from group "mpls-conf"
      ##
      retry-timer 15;
      ##
      ## "64k" was inherited from group "mpls-conf"
      ##
      bandwidth 64k;
      ##
      ## "120" was inherited from group "mpls-conf"
      ##
      optimize-timer 120;
    }
  }
}
```

Summary of Configuration Group Statements

The following sections explain each of the configuration group statements. The statements are organized alphabetically.

apply-groups

Syntax `apply-groups [group-name];`

Hierarchy Level All hierarchy levels

Description Apply a configuration group to a specific hierarchy level in a configuration, to have a configuration inherit the statements in the configuration group.

You can specify more than one group name. You must list them in order of inheritance priority. The configuration data in the first group takes priority over the data in subsequent groups.

For routers that support multiple Routing Engines, you can specify `re0` and `re1` as group names. The configuration specified in group `re0` is applied only if the current Routing Engine is in slot 0; likewise, the configuration specified in group `re1` is applied only if the current Routing Engine is in slot 1. Therefore, both Routing Engines can use the same configuration file, each using only the configuration statements that apply to it. Each `re0` or `re1` group contains at a minimum the configuration for the hostname and the management interface (`fxp0`). If each Routing Engine uses a different management interface, the group also should contain the configuration for the backup router and static routes.

You can include the `apply-groups` statement at any level of the configuration hierarchy.

You can include only one `apply-groups` statement at each specific level of the configuration hierarchy. The `apply-groups` statement at a specific hierarchy level lists the configuration groups to be added to the containing statement's list of configuration groups.

Options *group-name*—Names specified on the group statement.

Usage Guidelines See “Apply a Configuration Group” on page 230.

Required Privilege Level `configure`—To enter configuration mode; other required privilege levels depend on where the statement is located in the configuration hierarchy.

See Also `groups` on page 250

apply-groups-except

Syntax	<code>apply-groups-except [<i>group-name</i>];</code>
Hierarchy Level	All hierarchy levels except the top level
Description	Disables inheritance of a configuration group.
Options	<i>group-name</i> —Names specified on the group statement.
Usage Guidelines	See “Disable Inheritance of a Configuration Group” on page 232.
Required Privilege Level	configure—To enter configuration mode; other required privilege levels depend on where the statement is located in the configuration hierarchy.
See Also	groups on page 250

groups

Syntax	<pre>groups { <i>group-name</i> { <i>configuration-data</i>; } }</pre>
Hierarchy Level	[edit]
Description	Create a configuration group.
Options	<p><i>configuration-data</i>—The configuration statements that are to be applied elsewhere in the configuration with the <code>apply-groups</code> statement, to have the target configuration inherit the statements in the group.</p> <p><i>group-name</i>—Name of the configuration group. To configure multiple groups, specify more than one <i>group-name</i>. On routers that support multiple Routing Engines, you can also specify two special group names:</p> <p>re0—Configuration statements that are to be applied to the Routing Engine in slot 0.</p> <p>re1—Configuration statements that are to be applied to the Routing Engine in slot 1.</p>

The configuration specified in group re0 is applied only if the current Routing Engine is in slot 0; likewise, the configuration specified in group re1 is applied only if the current Routing Engine is in slot 1. Therefore, both Routing Engines can use the same configuration file, each using only the configuration statements that apply to it. Each re0 or re1 group contains at a minimum the configuration for the hostname and the management interface (fxp0). If each Routing Engine uses a different management interface, the group also should contain the configuration for the backup router and static routes.

Usage Guidelines See “Create a Configuration Group” on page 229.

Required Privilege Level configure—To enter configuration mode.

See Also apply-groups on page 249

