

Chapter 7

Managing Configurations

This chapter provides basic information about managing configurations.

Topics include:

- How the Configuration Is Stored on page 116
- Returning to the Most Recently Committed Configuration on page 117
- Returning to a Configuration Prior to the Most Recently Committed One on page 117
- Comparing Configuration Changes with a Prior Version on page 119
- Creating and Returning to a Rescue Configuration on page 120
- Saving a Configuration to a File on page 121
- Loading a Configuration from a File on page 123
- Synchronizing Routing Engines on page 132

How the Configuration Is Stored

When you edit a configuration, you work in a copy of the current configuration to create a candidate configuration. The changes you make to the candidate configuration are visible in the CLI immediately, so if multiple users are editing the configuration at the same time, all users can see all changes.

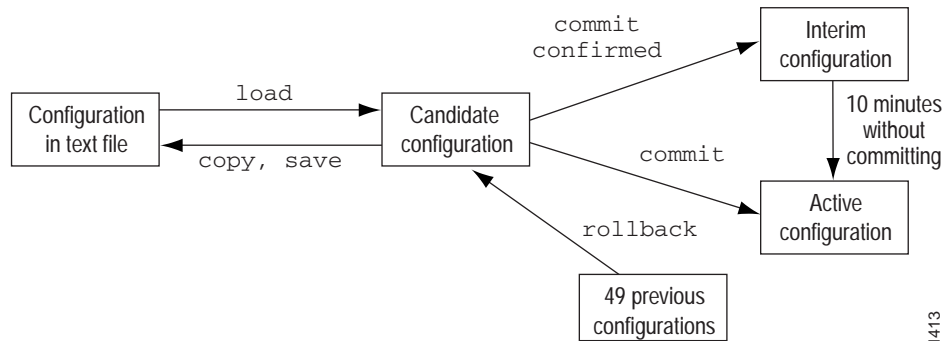
To have a candidate configuration take effect, you *commit* the changes. At this point, the candidate file is checked for proper syntax, activated, and marked as the current, operational software configuration file. If multiple users are editing the configuration, when you commit the candidate configuration, all changes made by all the users take effect.

In addition to saving the current configuration, the CLI saves the current operational version and the previous 49 versions of committed configurations. The most recently committed configuration is version 0 (the current operational version, which is the default configuration that the system returns to if you roll back to a previous configuration), and the oldest saved configuration is version 49.

The currently operational JUNOS software configuration is stored in the file `juniper.conf`, and the last three committed configurations are stored in the files `juniper.conf.1`, `juniper.conf.2`, and `juniper.conf.3`. These four files are located in the directory `/config`, which is on the router's flash drive. The remaining 46 previous versions of committed configurations, the files `juniper.conf.4` through `juniper.conf.49`, are stored in the directory `/var/db/config` on the hard disk.

Figure 16 illustrates the various router configuration states and the configuration mode commands you use to load, commit, copy, save, or roll back the configuration.

Figure 16: Commands for Storing and Modifying the Router Configuration



1413

Returning to the Most Recently Committed Configuration

To return to the most recently committed configuration and load it into configuration mode without activating it, use the `rollback` configuration mode command:

```
[edit]
user@host# rollback
```

Load complete

To activate the configuration to which you rolled back, use the `commit` command:

```
[edit]
user@host# rollback
load complete
[edit]
user@host# commit
```

Returning to a Configuration Prior to the Most Recently Committed One

To return to a configuration prior to the most recently committed one, include the number in the `rollback` command. *number* can be a number in the range 0 through 9. The most recently saved configuration is number 0 (which is the default configuration to which the system returns), and the oldest saved configuration is number 9.

```
[edit]
user@host# rollback number
```

Load complete

Displaying Previous Configurations

To display previous configurations, including the rollback number, date, time, the name of the user who committed changes, and the method of commit, use the `rollback ?` command.

```
[edit]
user@host# rollback ?
```

```
Possible completions:
<[Enter]> Execute this command
<number> Numeric argument
 0      2005-02-27 12:52:10 PST by abc via cli
 1      2005-02-26 14:47:42 PST by def via cli
 2      2005-02-14 21:55:45 PST by ghi via cli
 3      2005-02-10 16:11:30 PST by jkl via cli
 4      2005-02-10 16:02:35 PST by mno via cli
 5      2005-03-16 15:10:41 PST by pqr via cli
 6      2005-03-16 14:54:21 PST by stu via cli
 7      2005-03-16 14:51:38 PST by vwx via cli
 8      2005-03-16 14:43:29 PST by yzz via cli
 9      2005-03-16 14:15:37 PST by abc via cli
10      2005-03-16 14:13:57 PST by def via cli
11      2005-03-16 12:57:19 PST by root via other
12      2005-03-16 10:45:23 PST by root via other
```

```
13          2005-03-16 10:08:13 PST by root via other
14          2005-03-16 01:20:56 PST by root via other
15          2005-03-16 00:40:37 PST by ghi via cli
16          2005-03-16 00:39:29 PST by jkl via cli
17          2005-03-16 00:32:36 PST by mno via cli
18          2005-03-16 00:31:17 PST by pqr via cli
19          2005-03-15 19:59:00 PST by stu via cli
20          2005-03-15 19:53:39 PST by vwx via cli
21          2005-03-15 18:07:19 PST by yzz via cli
22          2005-03-15 17:59:03 PST by abc via cli
23          2005-03-15 15:05:14 PST by def via cli
24          2005-03-15 15:04:51 PST by ghi via cli
25          2005-03-15 15:03:42 PST by jkl via cli
26          2005-03-15 15:01:52 PST by mno via cli
27          2005-03-15 14:58:34 PST by pqr via cli
28          2005-03-15 13:09:37 PST by root via other
29          2005-03-12 11:01:20 PST by stu via cli
30          2005-03-12 10:57:35 PST by vwx via cli
31          2005-03-11 10:25:07 PST by yzz via cli
32          2005-03-10 23:40:58 PST by abc via cli
33          2005-03-10 23:40:38 PST by def via cli
34          2005-03-10 23:14:27 PST by ghi via cli
35          2005-03-10 23:10:16 PST by jkl via cli
36          2005-03-10 23:01:51 PST by mno via cli
37          2005-03-10 22:49:57 PST by pqr via cli
38          2005-03-10 22:24:07 PST by stu via cli
39          2005-03-10 22:20:14 PST by vwx via cli
40          2005-03-10 22:16:56 PST by yzz via cli
41          2005-03-10 22:16:41 PST by abc via cli
42          2005-03-10 20:44:00 PST by def via cli
43          2005-03-10 20:43:29 PST by ghi via cli
44          2005-03-10 20:39:14 PST by jkl via cli
45          2005-03-10 20:31:30 PST by root via other
46          2005-03-10 18:57:01 PST by mno via cli
47          2005-03-10 18:56:18 PST by pqr via cli
48          2005-03-10 18:47:49 PST by stu via cli
49          2005-03-10 18:47:34 PST by vw via cli
| Pipe through a command
[edit]
```

Comparing Configuration Changes with a Prior Version

In configuration mode only, when you have made changes to the configuration and want to compare the candidate configuration with a prior version, you can use the `compare` command to display the configuration. The `compare` command compares the candidate configuration with either the current committed configuration or a configuration file and displays the differences between the two configurations. To compare configurations, specify the `compare` command after the pipe:

```
[edit]
user@host# show | compare [filename | rollback n]
```

filename is the full path to a configuration file. The file must be in the proper format: a hierarchy of statements.

n is the index into the list of previously committed configurations. The most recently saved configuration is number 0, and the oldest saved configuration is number 49. If you do not specify arguments, the candidate configuration is compared against the active configuration file (`/config/juniper.conf`).

The comparison output uses the following conventions:

- Statements that are only in the candidate configuration are prefixed with a plus sign (+).
- Statements that are only in the comparison file are prefixed with a minus sign (-).
- Statements that are unchanged are prefixed with a single blank space ().

The following example shows various changes, then a comparison of the candidate configuration with the active configuration, showing only the changes made at the `[edit protocols bgp]` hierarchy level:

```
[edit]
user@host# edit protocols bgp

[edit protocols bgp]
user@host# show
group my-group {
    type internal;
    hold-time 60;
    advertise-inactive;
    allow 1.1.1.1/32;
}
group fred {
    type external;
    peer-as 33333;
    allow 2.2.2.2/32;
}
group test-peers {
    type external;
    allow 3.3.3.3/32;
}
[edit protocols bgp]
user@host# set group my-group hold-time 90
```

```

[edit protocols bgp]
user@host# delete group my-group advertise-inactive
[edit protocols bgp]
user@host# set group fred advertise-inactive
[edit protocols bgp]
user@host# delete group test-peers
[edit protocols bgp]
user@host# show | compare
[edit protocols bgp group my-group]
-   hold-time 60;
+   hold-time 90;
-   advertise-inactive;
[edit protocols bgp group fred]
+   advertise-inactive;
[edit protocols bgp]
-group test-peers {
-   type external;
-   allow 3.3.3.3/32;
-}
[edit protocols bgp]
user@host# show
group my-group {
    type internal;
    hold-time 90;
    allow 1.1.1.1/32;
}
group fred {
    type external;
    advertise-inactive;
    peer-as 3333;
    allow 2.2.2.2/32;
}

```

Creating and Returning to a Rescue Configuration

A *rescue* configuration allows you to define a known working configuration or a configuration with a known state that you can roll back to at any time. This alleviates the necessity of having to remember the rollback number with the `rollback` command. You use the rescue configuration when you need to roll back to a known configuration or as a last resort if your router configuration and the backup configuration files become damaged beyond repair.

To save the most recently committed configuration as the rescue configuration so that you can return to it at any time, issue the `request system configuration rescue save` command:

```

user@host> request system configuration rescue save
user@host>

```

To return to the rescue configuration, use the `rollback rescue` configuration mode command:

```
[edit]
user@host# rollback rescue
load complete
```

To activate the rescue configuration that you have loaded, use the `commit` command:

```
[edit]
user@host# rollback rescue
load complete
[edit]
user@host# commit
```

To save the most recently committed configuration as the rescue configuration so that you can return to it at any time using the `rollback` command, issue the `request system configuration rescue save` command:

```
user@host> request system configuration rescue save
user@host>
```

To delete an existing rescue configuration, issue the `request system configuration rescue delete` command:

```
user@host> request system configuration rescue delete
user@host>
```

For more information about the `request system configuration rescue delete` and `request system configuration rescue save` commands, see the *JUNOS System Basics and Services Command Reference*.

Saving a Configuration to a File

You might want to save the configuration to a file so that you can edit it with a text editor of your choice. You can save your current configuration to an ASCII file, which saves the configuration in its current form, including any uncommitted changes. If more than one user is modifying the configuration, all changes made by all users are saved.

To save software configuration changes to an ASCII file, use the `save` configuration mode command:

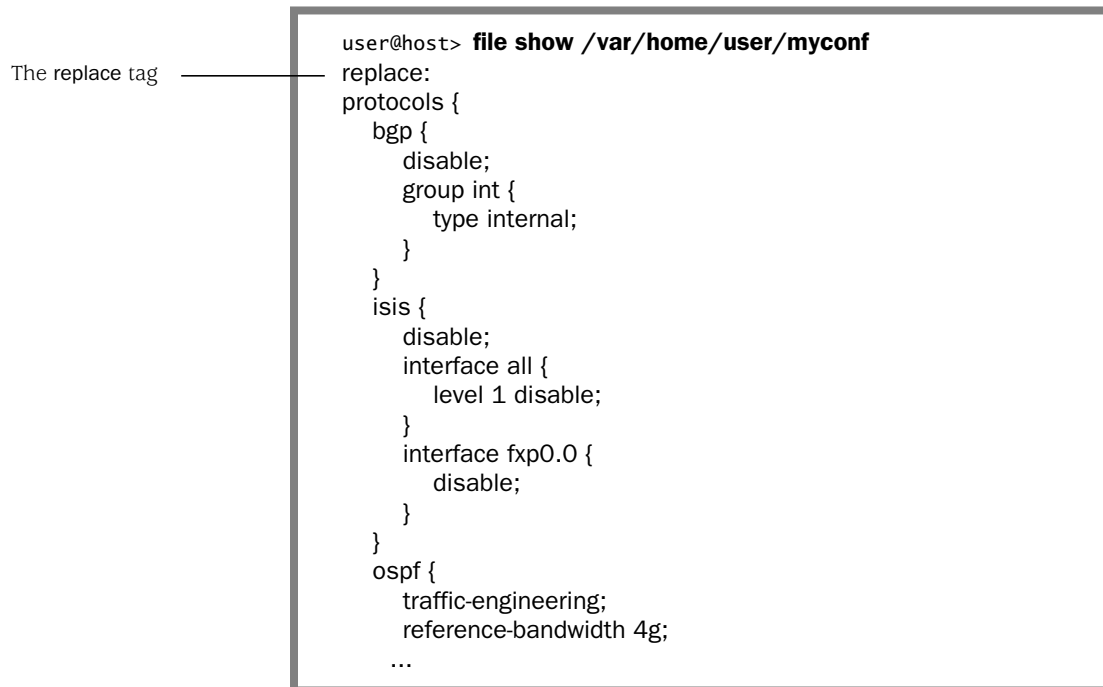
```
[edit]
user@host# save filename
[edit]
user@host#
```

The contents of the current level of the statement hierarchy (and below) are saved, along with the statement hierarchy containing it. This allows a section of the configuration to be saved, while fully specifying the statement hierarchy.

By default, the configuration is saved to a file in your home directory, which is on the flash drive. For information about specifying the filename, see “Specifying Filenames and URLs” on page 64.

When you issue this command from anywhere in the hierarchy (except the top level), a **replace** tag is automatically included at the beginning of the file. You can use the **replace** tag to control how a configuration is loaded from a file. (See Figure 17.) For more information, see “Loading a Configuration from a File” on page 123.

Figure 17: The replace Tag



```
user@host> file show /var/home/user/myconf
replace:
protocols {
  bgp {
    disable;
    group int {
      type internal;
    }
  }
  isis {
    disable;
    interface all {
      level 1 disable;
    }
    interface fxp0.0 {
      disable;
    }
  }
  ospf {
    traffic-engineering;
    reference-bandwidth 4g;
    ...
```

Loading a Configuration from a File

You can create a file, copy the file to the local router, and then load the file into the CLI. After you have loaded the file, you can commit it to activate the configuration on the router, or you can edit the configuration interactively using the CLI and commit it at a later time.

You can also create a configuration while typing at the terminal and then load it. Loading a configuration from the terminal is generally useful when you are cutting existing portions of the configuration and pasting them elsewhere in the configuration.

To load an existing configuration file that is located on the router, use the `load` configuration mode command:

```
[edit]
user@host# load (merge | override | patch | replace | set | update) filename
<relative>
```

To load a configuration from the terminal, use the following version of the `load` configuration mode command:

```
[edit]
user@host# load (merge | override | patch | replace | set | update) terminal
<relative>
[Type ^D to end input]
```

To replace an entire configuration, specify the `override` option at any level of the hierarchy.

An `override` operation discards the current candidate configuration and loads the configuration in *filename* or the one that you type at the terminal. When you use the `override` option and commit the configuration, all system processes reparse the configuration. For an example, see Figure 18 on page 126.

To replace only the configuration that has changed, specify the `update` option at any level of the hierarchy. An `update` operation compares the current configuration and the current candidate configuration, and loads only the changes between these configurations in *filename* or the one that you type at the terminal. When you use the `update` operation and commit the configuration, the JUNOS software attempts to notify the smallest set of system processes that are affected by the configuration change.

To combine the current configuration and the configuration in *filename* or the one that you type at the terminal, specify the `merge` option. A `merge` operation is useful when you are adding a new section to an existing configuration. If the existing configuration and the incoming configuration contain conflicting statements, the statements in the incoming configuration override those in the existing configuration. For an example, see Figure 20 on page 127.

To replace portions of a configuration, specify the **replace** option. For this operation to work, you must include **replace:** tags in the file or configuration you type at the terminal. The software searches for the **replace:** tags, deletes the existing statements of the same name, if any, and replaces them with the incoming configuration. If there is no existing statement of the same name, the **replace** operation adds to the configuration the statements marked with the **replace:** tag. For an example, see Figure 19 on page 126.

To load a configuration that contains the **set** configuration mode command, specify the **set** option. This option executes the configuration instructions line by line as they are stored in a file or from a terminal. The instructions can contain any configuration mode command, such as **set**, **edit**, **exit**, and **top**. For an example, see Figure 22 on page 128.

To use the **merge**, **replace**, **set**, or **update** option without specifying the full hierarchy level, specify the **relative** option. For example:

```
[edit system]
user@host# show static-host-mapping
bob sysid 987.654.321ab

[edit system]
user@host# load replace terminal relative
{Type ^D at a new line to end input]
replace: static-host-mapping {
    bob sysid 0123.456.789bc;
}
load complete

[edit system]
user@host# show static-host-mapping
bob sysid 0123.456.789bc;
```

To change part of the configuration with a patch file and mark only those parts as changed, specify the **patch** option. For an example, see Figure 21 on page 127.

If, in an override or merge operation, you specify a file or type text that contains **replace:** tags, the **replace:** tags are ignored, and the override or merge operation is performed.

If you are performing a **replace** operation and the file you specify or text you type does not contain any **replace:** tags, the replace operation is effectively equivalent to a **merge** operation. This might be useful if you are running automated scripts and cannot know in advance whether the scripts need to perform a replace or a merge operation. The scripts can use the **replace** operation to cover either case.

For information about specifying the filename, see “Specifying Filenames and URLs” on page 64.

To copy a configuration file from another network system to the local router, you can use the SSH and Telnet utilities, as described in the *JUNOS System Basics and Services Command Reference*.



NOTE: If you are using JUNOS software in a Common Criteria environment, system log messages are created whenever a **secret** attribute is changed (for example, password changes or changes to the RADIUS shared secret). These changes are logged during the following configuration load operations:

```
load merge
load replace
load override
load update
```

For more information, see the *Secure Configuration Guide for Common Criteria and JUNOS-FIPS*.

Examples: Loading a Configuration from a File

Figure 18: Example 1: Load a Configuration from a File

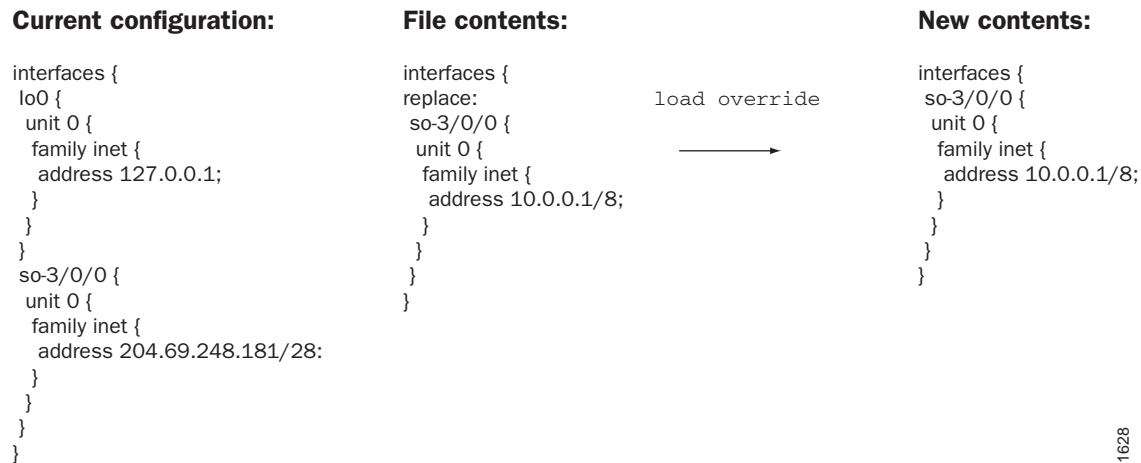


Figure 19: Example 2: Load a Configuration from a File

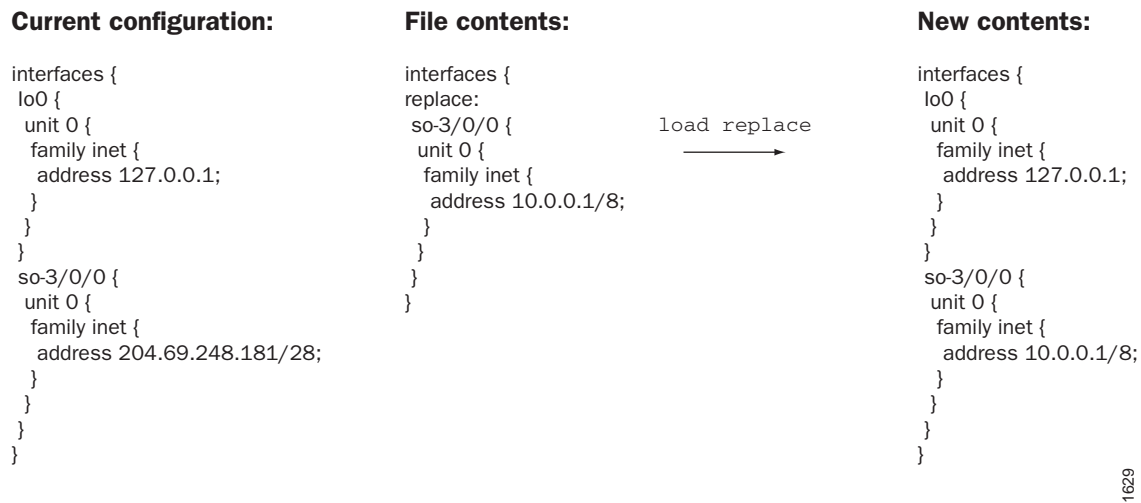


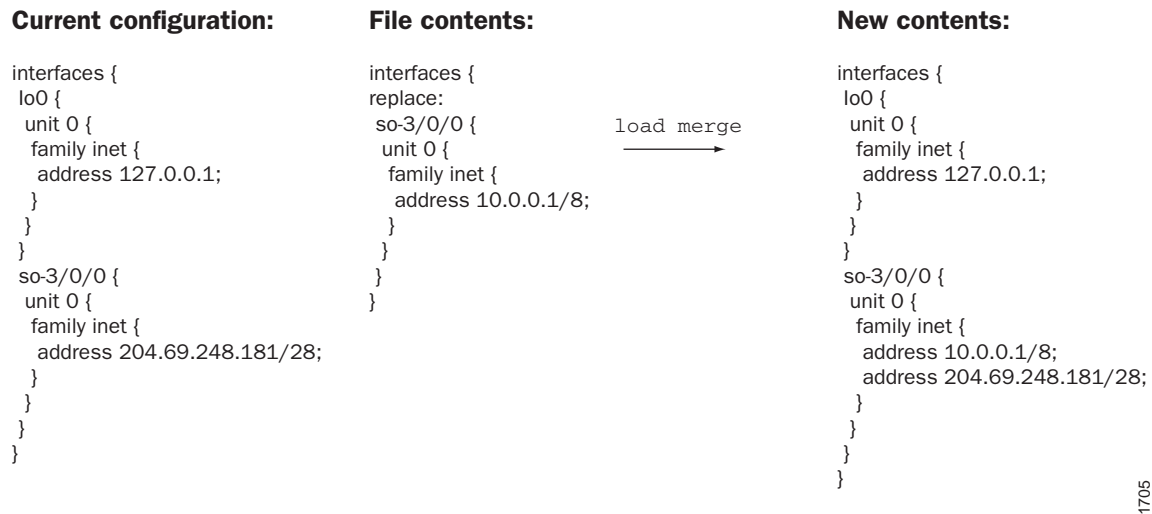
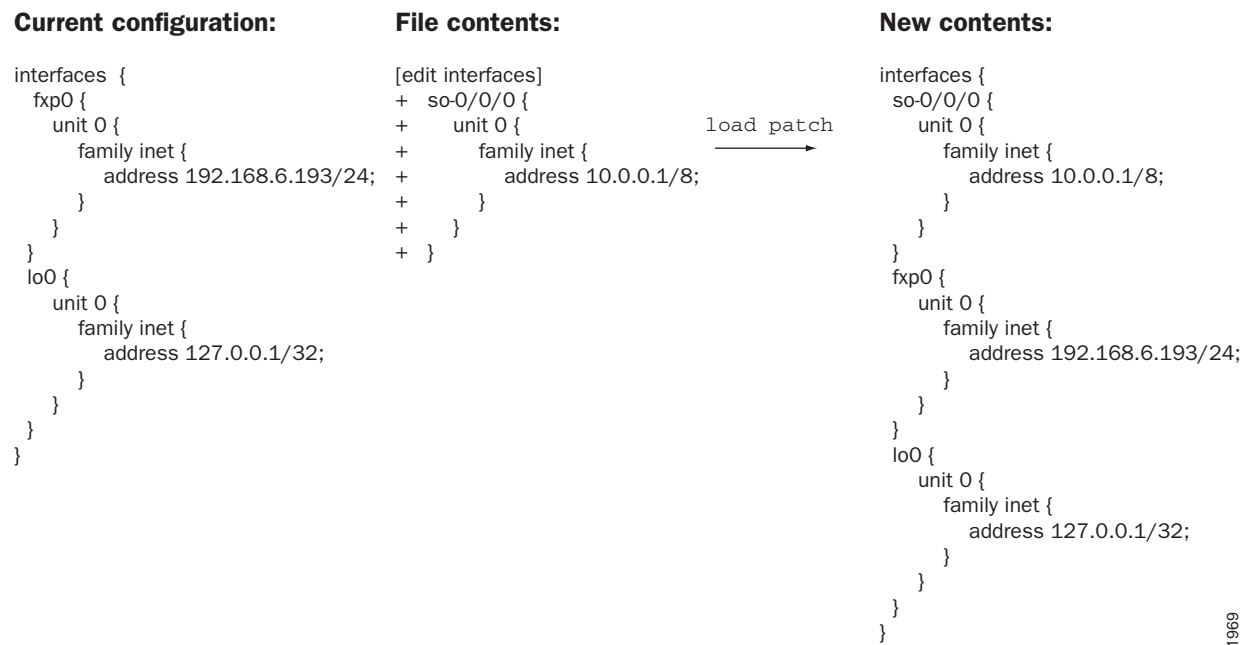
Figure 20: Example 3: Load a Configuration from a File**Figure 21: Example 4: Load a Configuration from a File**

Figure 22: Example 5: Load a Configuration from a File**File contents:**

```
edit access
set profile p1 client cl ike
edit profile p1 client cl ike
set pre-shared-key ascii-text "abcd"
set allowed-proxy-pair local 1.1.1.1 remote 2.2.2.2
exit
deactivate profile p1
top
edit system
set radius-server 1.1.1.1
```

```
load set
```

**New contents:**

```
system {
  radius-server {
    1.1.1.1;
  }
}
access {
  inactive: profile p1 {
    client cl {
      ike {
        allowed-proxy-pair local 1.1.1.1/32 remote 2.2.2.2/32;
        pre-shared-key ascii-text " 9 dg4 Djqf5FVw"; ## SECRET-DATA
      }
    }
  }
}
}
```

Additional Details About Specifying Statements and Identifiers

This section provides more detailed information about CLI container and leaf statements so that you can better understand how you must specify them when creating ASCII configuration files. It also describes how the CLI performs type-checking to verify that the data you entered is in the correct format.

Specifying Statements

Statements are shown one of two ways, either with braces or without:

- Statement name and identifier, with one or more lower-level statements enclosed in braces:

```
< statement-name > < identifier > {
  statement;
  additional-statements;
}
```

- Statement name, identifier, and a single identifier:

`< statement-name > < identifier > identifier;`

The *statement-name* is the name of the statement. In the configuration example shown in the previous section, `ospf` and `area` are statement names.

The *identifier* is a name or other string that uniquely identifies an instance of a statement. The identifier is used when a statement can be specified more than once in a configuration.

When specifying a statement, you must specify either a statement name or an identifier, or both, depending on the statement hierarchy.

You specify identifiers in one of the following ways:

- *identifier*—The *identifier* is a flag, which is a single keyword.
- *identifier value*—The *identifier* is a keyword, and the *value* is a required option variable.
- *identifier [value1 value 2 value3 ...]*—The *identifier* is a set that accepts multiple values. The brackets are required when you specify a set of identifiers; however, they are optional when you specify only one identifier.

The following examples illustrate how statements and identifiers are specified in the configuration:

```

protocol {
    ospf {
        area 0.0.0.0 {
            interface so-0/0/0 {# which contains an interface named "so-0/0/0."
                hello-interval 25;# Identifier and value (identifier-name value).
                priority 2;      # Identifier and value (identifier-name value).
                disable;        # Flag identifier (identifier-name).
            }
            interface so-0/0/1;# Another instance of "interface," named so-0/0/1,
        }
    }
}
policy-options {
    term term1 {
        from {
            route-filter 10.0.0.0/8 orlonger reject;# One identifier ("route-filter") with
            route-filter 127.0.0.0/8 orlonger reject;# multiple values.
            route-filter 128.0.0.0/16 orlonger reject;
            route-filter 149.20.64.0/24 orlonger reject;
            route-filter 172.16.0.0/12 orlonger reject;
            route-filter 191.255.0.0/16 orlonger reject;
        }
        then {
            next term;
        }
    }
}

```

When you create an ASCII configuration file, you can specify statements and identifiers in one of the following ways. However, each statement has a preferred style, and the CLI uses that style when displaying the configuration in response to a configuration mode `show` command.

- Statement followed by identifiers:

```
statement-name identifier-name [...] identifier-name value [...];
```

- Statement followed by identifiers enclosed in braces:

```
statement-name {
    identifier-name;
    [...]
    identifier-name value;
    [...]
}
```

- For some repeating identifiers, you can use one set of braces for all the statements:

```
statement-name {
    identifier-name value1;
    identifier-name value2;
}
```

Performing CLI Type-Checking

When you specify identifiers and values, the CLI expects to receive specific types of input and performs type-checking to verify that the data you entered is in the correct format. For example, for a statement in which you must specify an IP address, the CLI checks that you entered an address in a valid format. If you have not, an error message indicates what you were expected to type. Table 14 lists the data types the CLI checks.

Table 14: CLI Configuration Input Types (1 of 2)

Data Type	Format	Examples
Physical interface name (used in the [edit interfaces] hierarchy)	<i>type-fpc/pic/port</i>	Correct: so-0/0/1 Incorrect: so-0
Full interface name	<i>type-fpc/pic/port<:channel>.logical</i>	Correct: so-0/0/1.0 Incorrect: so-0/0/1
Full or abbreviated interface name (used in places other than the [edit interfaces] hierarchy)	<i>type-<fpc/>/pic/port>><<:channel>.logical></i>	Correct: so, so-1, so-1/2/3:4.5
IP address	<i>0xhex-bytes</i> <i>octet<.octet<.octet.<octet>>></i>	Correct: 1.2.3.4, 0x01020304, 128.8.1, 128.8 Sample translations: 1.2.3 becomes 1.2.3.0 0x01020304 becomes 1.2.3.4 0x010203 becomes 0.1.2.3

Table 14: CLI Configuration Input Types (2 of 2)

Data Type	Format	Examples
IP address (destination prefix) and prefix length	<i>O</i> hex-bytes</length> <i>o</i> ctet<. <i>o</i> ctet<. <i>o</i> ctet.< <i>o</i> ctet>>></length>	Correct: 10/8, 128.8/16, 1.2.3.4/32, 1.2.3.4 Sample translations: 1.2.3 becomes 1.2.3.0/32 0x01020304 becomes 1.2.3.4/32 0x010203 becomes 0.1.2.3/32 default becomes 0.0.0.0/0
International Organization for Standardization (ISO) address	<i>h</i> ex-nibble< <i>h</i> ex-nibble ...>	Correct: 47.1234.2345.3456.00, 47123423453456.00, 47.12.34.23.45.34.56.00 Sample translations: 47123456 becomes 47.1234.56 47.12.34.56 becomes 47.1234.56 4712.3456 becomes 47.1234.56
OSPF area identifier (ID)	<i>O</i> hex-bytes <i>o</i> ctet<. <i>o</i> ctet<. <i>o</i> ctet.< <i>o</i> ctet>>> <i>d</i> ecimal-number	Correct: 54, 0.0.0.54, 0x01020304, 1.2.3.4 Sample translations: 54 becomes 0.0.0.54 257 becomes 0.0.1.1 128.8 becomes 128.8.0.0 0x010203 becomes 0.1.2.3

Synchronizing Routing Engines

If your router has two Routing Engines, you can manually direct one Routing Engine to synchronize its configuration with the other by issuing the **commit synchronize** command. The Routing Engine on which you execute this command (requesting Routing Engine) copies and loads its candidate configuration to the other (responding Routing Engine). Both Routing Engines then perform a syntax check on the candidate configuration file being committed. If no errors are found, the configuration is activated and becomes the current operational configuration on both Routing Engines. The **commit synchronize** command does not work if the responding Routing Engine has uncommitted configuration changes.

For example, if you are logged in to **re1** (requesting Routing Engine) and you want **re0** (responding Routing Engine) to have the same configuration as **re1**, issue the **commit synchronize** command on **re1**. **re1** copies and loads its candidate configuration to **re0**. Both Routing Engines then perform a syntax check on the candidate configuration file being committed. If no errors are found, **re1**'s candidate configuration is activated and becomes the current operational configuration on both Routing Engines.



NOTE: When you issue the **commit synchronize** command, you must use the groups **re0** and **re1**. For information about how to use the **apply groups** statement, see “Applying a Configuration Group” on page 165.

The responding Routing Engine must be running JUNOS Release 5.0 or higher.

For information about issuing the **commit synchronize** command on a routing matrix, see the *JUNOS System Basics Configuration Guide*.

To synchronize a Routing Engine's current operational configuration file with the other, log in to the Routing Engine from which you want to synchronize and issue the **commit synchronize** command:

```
[edit]
user@host# commit synchronize
commit complete
[edit]
user@host#
```



NOTE: You can also add the **commit synchronize** statement at the **[edit system]** hierarchy level so that a **commit** command automatically invokes a **commit synchronize** command by default. For more information, see the *JUNOS System Basics Configuration Guide*.

Example: Using Apply Groups re0 and re1

The following example shows apply groups re0 and re1 with some configuration data that might be different on re0 and re1:

```

re0 {
  system {
    host-name my_router_RE0;
  }
  interfaces {
    fxp0 {
      unit 0 {
        family inet {
          address 192.168.15.49/24;
        }
        family iso;
      }
    }
  }
}
re1 {
  system {
    host-name my_router_RE1;
  }
  interfaces {
    fxp0 {
      unit 0 {
        family inet {
          address 192.168.15.50/24;
        }
        family iso;
      }
    }
  }
}

```

Example: Setting Apply Groups re0 and re1

The following example sets the apply groups re0 and re1:

```

[edit]
user@host# set apply-groups [re0 re1]
[edit]
user@host#

```

