JUNIPER NETWORKS

# Junos® OS

## Configuration Automation

Release
11.4

Published: 2011-11-08
Revision 1

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net



*Junos® OS Configuration Automation*


Revision History
October 2011—Revision 1; initial release

The information in this document is current as of the date listed in the revision history.

## END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at http://www.juniper.net/support/eula.html. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

# Table of Contents

# List of Figures

# List of Tables

PART 1

# Overview

CHAPTER 1

# Commit Scripts Overview



## Commit Script Overview

Junos OS commit scripts enforce custom configuration rules during the commit process. When a candidate configuration is committed, it is inspected by each active commit script. If a configuration violates your custom rules, the script can instruct Junos OS to take appropriate action. A commit script can perform the following actions:

- Generate and display custom warning messages to the user
- Generate and log custom system log (syslog) messages
- Change the configuration to conform to the custom business rules
- Generate a commit error and halt the commit operation

Commit scripts are based on the Junos XML management protocol and the Junos XML API. The Junos XML management protocol is an XML based RPC mechanism, and the Junos XML API is an XML representation of Junos configuration statements and operational mode commands.

Commit scripts can be written in either the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) scripting language. Commit scripts use the XML Path Language (XPath) to locate the configuration objects to be inspected and XSLT or SLAX constructs to specify the actions to perform on the configuration objects. The actions can change the configuration or generate messages about it. For more information about XSLT, see XSLT Overview. For more information about SLAX, see SLAX Overview

Additionally, you can create *macros*, which allow you to create custom configuration syntax that simplifies the task of configuring a device running Junos OS. By itself, your custom syntax has no operational impact on the device. A corresponding commit script macro uses your custom syntax as input data for generating standard Junos OS configuration statements that execute your intended operational impact.

To view the device's current configuration in the Extensible Markup Language (XML), using the command-line interface's (CLI's) operational mode, issue the **show configuration | display xml** command. To view your configuration in commit-script-style XML, issue the **show configuration | display commit-scripts view** command. Commit-script-style XML view displays the configuration in the format that would be input to a commit script.

**Related Documentation**

- Junos XML API and Junos XML Management Protocol Overview
- SLAX Overview
- XSLT Overview

## Advantages of Using Commit Scripts

Reducing human error in a network configuration can significantly improve network uptime. Commit scripts enable you to control operational practices and enforce operational policy, thereby decreasing the possibility of human error. Restricting device configurations in accordance with custom design rules can vastly improve network reliability.

Consider the following examples of actions you can perform with commit scripts:

- Basic sanity test—Ensure that the **[edit interfaces]** and **[edit protocols]** hierarchies have not been accidentally deleted.
- Consistency check—Ensure that every T1 interface configured at the **[edit interfaces]** hierarchy level is also configured at the **[edit protocols rip]** hierarchy level.
- Dual Routing Engine configuration test—Ensure that the **re0** and **re1** configuration groups are set up correctly. When you use configuration groups, the inherited values can be overridden in the target configuration. A commit script can determine if an individual target configuration element is blocking proper inheritance of the configuration group settings.
- Interface density—Ensure that a channelized interface does not have too many channels configured.
- Link scaling—Ensure that SONET/SDH interfaces never have a maximum transmission unit (MTU) size less than 4 kilobytes (KB).
- Import policy check—Ensure that an interior gateway protocol (IGP) does not use an import policy that imports the full routing table.
- Cross-protocol checks—Ensure that all LDP-enabled interfaces are configured for an IGP, or ensure that all IGP-enabled interfaces are configured for LDP.
- IGP design check—Ensure that Level 1 IS-IS routers are never enabled.

When a candidate configuration does not adhere to your design rules, a commit script can instruct Junos OS to generate custom warnings, system log messages, or error messages that block the commit operation from succeeding. In addition, the commit script can change the configuration in accordance with your rules and then proceed with the commit operation.

Consider a network design that requires every interface on which the International Organization for Standardization (ISO) family of protocols is enabled to also have MPLS enabled. At commit time, a commit script inspects the configuration and issues an error if this requirement is not met. This error causes the commit operation to fail and forces the user to update the configuration to comply.

Instead of an error, the commit script can issue a warning about the configuration problem and then automatically correct it by changing the configuration to enable MPLS on all interfaces. A system log message can also be generated, indicating that corrective action was taken.

Another option is to define a macro that enables ISO protocols and MPLS when the macro is applied to an interface. Configuring this macro simplifies the configuration task while ensuring that both protocols are configured together.

Finally, you can have the commit script correct the configuration using a *transient change*. In our example, a transient change allows MPLS to always be enabled on ISO-enabled interfaces without having the configuration statements appear in the candidate configuration.

NOTE: Transient changes cause a change to be generated in the *checkout configuration* but not in the candidate configuration. The checkout configuration is the configuration database that is checked for standard Junos OS syntax just before a configuration becomes active. This means transient changes are not saved in the configuration if the associated commit script is deleted or deactivated. The **show configuration | display commit-scripts** command displays all the statements that are in the configuration, including statements that were generated by transient changes. For more information, see "Overview of Generating Persistent or Transient Configuration Changes" on page 13.

**Related Documentation**

- Commit Script Overview on page 3

## How Commit Scripts Work

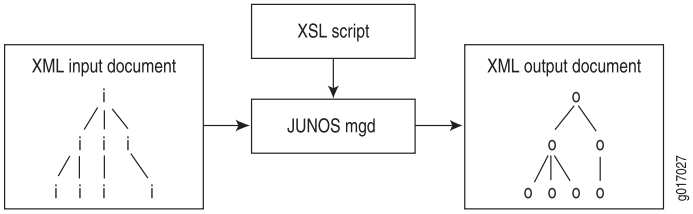
You enable commit scripts by listing the names of one or more commit script files at the **[edit system scripts commit]** hierarchy level. These scripts contain instructions that enforce custom configuration rules. Commit scripts are invoked during the commit process before the standard Junos OS validity checks are performed.

When you perform a commit operation, Junos OS executes each script in turn, passing the information in the candidate configuration to the scripts. The script inspects the configuration, performs the necessary tests and validations, and generates a set of instructions for performing certain actions. These actions include generating error, warning, and system log messages. If errors are generated, the commit operation fails and the candidate configuration remains unchanged. This is the same behavior that occurs with standard commit errors.

Commit scripts can also generate changes to the system configuration. Because the changes are loaded before the standard validation checks are performed, they are validated for correct syntax, just like statements already present in the configuration before the script is applied. If the syntax is correct, the configuration is activated and becomes the active, operational device configuration.

Figure 1 on page 6 shows the flow of commit script input and output.

### Figure 1: Commit Script Input and Output




Commit scripts cannot make configuration changes to protected statements or within protected hierarchies. If a commit script attempts to modify or delete a protected statement or hierarchy, Junos OS issues a warning that the change cannot be made. Failure to modify a protected configuration element does not halt the commit script or the commit process.

The following sections discuss several important concepts related to the commit script input and output:

- Commit Script Input on page 6
- Commit Script Output on page 7
- Commit Scripts and the Junos OS Commit Model on page 8

## Commit Script Input

The input for a commit script is the postinheritance candidate configuration in Junos XML API format. The term *postinheritance* means that all configuration group values have been inherited by their targets in the candidate configuration and the inactive portions of the configuration have been removed. For more information about configuration groups, see the *Junos OS CLI User Guide*.

When you issue the **commit** command, Junos OS automatically generates the candidate configuration in XML format and reads it into the management (mgd) process, at which time the input is evaluated by any commit scripts.

To display the XML format of the postinheritance configuration, issue the **show | display commit-scripts view** command:

```
[edit]
user@host# show | display commit-scripts view
```

To display all configuration groups data, including script-generated changes to the groups, issue the **show groups | display commit-scripts** command:

```
[edit]
user@host# show groups | display commit-scripts
```

To save the commit script input to a file, add the **save** command to the command line:

```
[edit]
user@host# show | display commit-scripts view | save filename.xml
```

By default, the file is placed in your home directory on the switch, router, or security device.

## Commit Script Output

To specify the desired commit script output—including warning, error, and system log messages, persistent changes, and transient changes—the script can contain tags that appear in any order, in any number. The tags for specifying output are as follows:

- **<xnm:warning>**—Generates a warning message
- **<xnm:error>**—Generates an error message.
- **<syslog><message>**—Generates a system log message.
- **<change>**—Generates a persistent change to the configuration.
- **<transient-change>**—Generates a transient change to the configuration.
- **<xsl:call-template name="jcs:emit-change">**
  **<xsl:with-param name="content">**—Generates a persistent change relative to the current context node as defined by an XPath expression.
- **<xsl:call-template name="jcs:emit-change">**
  **<xsl:with-param name="tag" select="'transient-change'"/>**
  **<xsl:with-param name="content">**—Generates a transient change relative to the current context node as defined by an XPath expression.
- **<xsl:call-template name="jcs:emit-change">**
  **<xsl:with-param name="message">**
  **<xsl:text>**—Generates a warning message in conjunction with a configuration change. You can use this set of tags to generate a notification that the configuration has been changed.

Junos OS processes this output and performs the appropriate actions. Errors and warnings are passed back to the Junos OS CLI or to a Junos XML protocol client application. The presence of an error automatically causes the commit operation to fail. Persistent and transient changes are loaded into the appropriate configuration database.

To test the output of error, warning, and system log messages from commit scripts, issue the **commit check | display xml** command:

```
[edit]
user@host# commit check | display xml
```

To display a detailed trace of commit script processing, issue the **commit check | display detail** command:

```
[edit]
user@host# commit check | display detail
```

NOTE: System log messages do not appear in the trace output, so you cannot use the commit check operation to test script-generated system log messages. Furthermore, system log messages are written to the system log during a commit operation, but not during a commit check operation.

**Related Documentation**

- Example: Protecting the Junos OS Configuration from Modification or Deletion.
- jcs:emit-change Template

## Commit Scripts and the Junos OS Commit Model

Junos OS uses a commit model to update the device's configuration. This model allows you to make a series of changes to a candidate configuration without affecting the operation of the device. When the changes are complete, you can commit the configuration. The commit operation saves the candidate configuration changes into the current configuration.

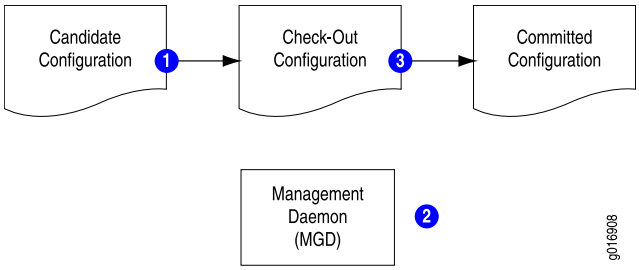
When you commit a set of changes in the candidate configuration, two methods are used to forward these changes to the current configuration:

- Standard commit model—Used when no commit scripts are active on the device.
- Commit script model—Incorporates commit scripts into the commit model.

### Standard Commit Model

In the standard commit model, the management (mgd) process validates the candidate configuration based on standard Junos validation rules. If the configuration file is valid, it becomes the current active configuration. Figure 2 on page 8 and the accompanying discussion explain how the standard commit model works:

**Figure 2: Standard Commit Model**



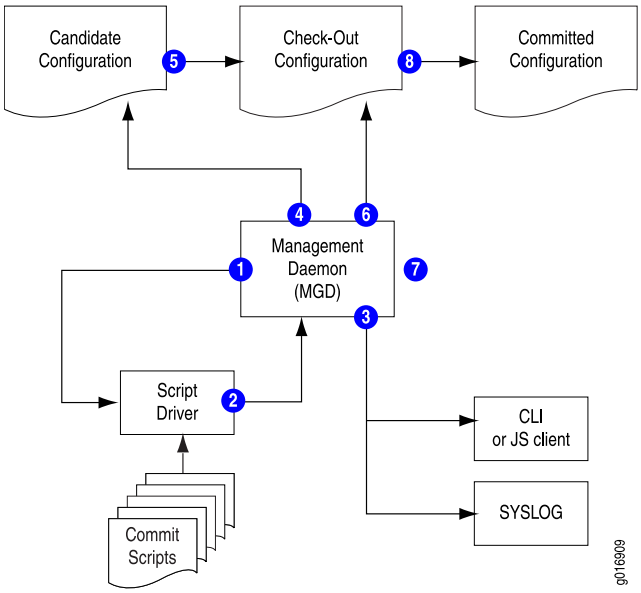

In the standard commit model, the software performs the following steps:

1. When the candidate configuration is committed, it is copied to become the checkout configuration.
2. The mgd process validates the checkout configuration.
3. If no error occurs, the checkout configuration is copied as the current active configuration.

## Commit Model with Commit Scripts

When commit scripts are added to the standard commit model, the process becomes more complex. The mgd process first passes an XML-formatted checkout configuration to a script driver, which handles the verification of the checkout configuration by the commit scripts. When verification is complete, the script driver returns an XML *action file* to the mgd process. The mgd process follows the instructions in the action file to update the candidate and checkout configurations, issue messages to the CLI, and write information to the system log as required. After processing the action file, the mgd process performs the standard Junos OS validation. Figure 3 on page 9 and the accompanying discussion explain this process.

### Figure 3: Commit Model with Commit Scripts Added




In the commit script model, Junos OS performs the following steps:

1. When the candidate configuration is committed, the mgd process sends the XML-formatted candidate configuration to the script driver.
2. Each enabled commit script is invoked against the candidate configuration, and each script can generate a set of actions for the mgd process to perform. The actions are collected in an XML action file.
3. The mgd process performs the following actions in response to **<error>**, **<warning>**, and **<syslog>** tag elements in the action file:
   - **<error>**—The mgd process halts the commit process (that is, the commit operation fails), returns an error message to the CLI or Junos XML protocol client, and takes no further action.
   - **<warning>**—The mgd process forwards the message to the CLI or the Junos XML protocol client.
   - **<syslog>**—The mgd process forwards the message to the system log process.

4. If the action file includes any **<change>** tag elements, the mgd process loads the requested changes into the candidate configuration.
5. The candidate configuration is copied to become the checkout configuration.
6. If the action file includes any **<transient-change>** tag elements, the mgd process loads the requested changes into the checkout configuration.
7. The mgd process validates the checkout configuration.
8. If there are no validation errors, the checkout configuration is copied to become the current active configuration.

NOTE: **Commit scripts cannot make configuration changes to protected statements or within protected hierarchies. If a commit script attempts to modify or delete a protected statement or hierarchy, Junos OS issues a warning that the change cannot be made. Failure to modify a protected configuration element does not halt the commit script or the commit process.**

Changes that are made to the candidate configuration during the commit operation are not evaluated by the custom rules during that commit operation. However, persistent changes are maintained in the candidate configuration and are evaluated by the custom rules during subsequent commit operations. For more information about how commit scripts change the candidate configuration, see "Avoiding Potential Conflicts When Using Multiple Commit Scripts" on page 29.

Transient changes are never evaluated by the custom rules in commit scripts, because they are made to the checkout configuration only after the commit scripts have evaluated the candidate configuration and the candidate is copied to become the checkout configuration. To remove a transient change from the configuration, remove, disable, or deactivate the commit script (as discussed in "Controlling Execution of Commit Scripts During Commit Operations" on page 38), or comment out the code that generates the transient change.

For more information about differences between persistent and transient changes, see "Overview of Generating Persistent or Transient Configuration Changes" on page 13.

**Related Documentation**

- Avoiding Potential Conflicts When Using Multiple Commit Scripts on page 29

CHAPTER 2

# Generating a Custom Warning, Error, or System Log Message



## Overview of Generating Custom Warning, Error, and System Log Messages

You can use a commit script to specify configuration rules that you always want to enforce. If a rule is broken, the commit script can emit a warning, error, or system log message.

In the Junos OS command-line interface (CLI), warning messages are emitted during commit operations to alert you that the configuration is not complete or contains a syntax error. If a custom configuration rule is broken, a custom warning message notifies you about the problem. The commit script causes the warning message to be passed back to the Junos OS CLI or to a Junos XML protocol client application. Unlike error messages, warning messages do not cause the commit operation to fail, so they are used for configuration problems that do not affect network traffic. A warning is best used as a response to configuration settings that do not adhere to recommended practices. An example of this type of configuration setting might be assignment of the same user ID to different users.

Alternatively, you can generate a custom warning message for a serious configuration problem, and specify an automatic configuration change that rectifies the problem. For more information about the use of warning messages in conjunction with automatic configuration changes, see "Overview of Generating Persistent or Transient Configuration Changes" on page 13.

Unlike warning messages, a custom error message causes the commit operation to fail and notifies the user about the configuration problem. The commit script causes the error message to be passed back to the Junos OS CLI or to a Junos XML protocol client application. Because error messages cause the commit operation to fail, they are used for problems that affect network traffic. An error message is best used as a response to configuration settings that you want to disallow—for example, when required statements are omitted from the configuration.

Junos OS generates system log messages (also called syslog messages) to record events that occur on the device, including the following:

- Routine operations, such as creation of an OSPF protocol adjacency or a user login into the configuration database
- Failure and error conditions, such as failure to access a configuration file or unexpected closure of a connection to a child or peer process
- Emergency or critical conditions, such as device power-down due to excessive temperature

Each system log message identifies the Junos OS process that generated the message and briefly describes the operation or error that occurred. The *Junos OS System Log Messages Reference* provides more detailed information about system log messages.

With commit scripts, you can cause custom system log messages to be generated in response to particular events that you define. For example, if a configuration rule is broken, a custom message can be generated to record this occurrence. If the commit script corrects the configuration, a custom message can indicate that corrective action was taken.

**Related Documentation**

- Example: Generating a Custom Error Message on page 51
- Example: Generating a Custom System Log Message on page 54
- Example: Generating a Custom Warning Message on page 48
- Generating a Custom Warning, Error, or System Log Message on page 43
- Tag Elements to Use When Generating Messages on page 46

CHAPTER 3

# Generating a Persistent or Transient Configuration Changes



## Overview of Generating Persistent or Transient Configuration Changes

Junos OS commit scripts enforce custom configuration rules. When a candidate configuration includes statements that you have decided must not be included in your configuration, or when the candidate configuration omits statements that you have decided are required, commit scripts can automatically change the configuration and thereby correct the problem.



### Differences Between Persistent and Transient Changes

Configuration changes made by commit scripts can be *persistent* or *transient*.

A persistent change remains in the candidate configuration and affects routing operations until you explicitly delete it, even if you subsequently remove or disable the commit script that generated the change and reissue the **commit** command. In other words, removing the commit script does not cause a persistent change to be removed from the configuration.

A transient change, in contrast, is made in the *checkout configuration* but not in the candidate configuration. The checkout configuration is the configuration database that is inspected for standard Junos OS syntax just before it is copied to become the active configuration on the device. If you subsequently remove or disable the commit script that made the change and reissue the **commit** command, the change is no longer made to the checkout configuration and so does not affect the active configuration. In other words, removing the commit script effectively removes a transient change from the configuration.

A common use for transient changes is to eliminate the need to repeatedly configure and display well-known policies, thus allowing these policies to be enforced implicitly. For example, if MPLS must be enabled on every interface with an International

Organization for Standardization (ISO) protocol enabled, the change can be transient, so that the repetitive or redundant configuration data need not be carried or displayed in the candidate configuration. Furthermore, transient changes allow you to write script instructions that apply the change only if a set of conditions is met.

Persistent and transient changes are loaded into the configuration in the same manner that the **load replace** configuration mode command loads an incoming configuration. When generating a persistent or transient change, adding the **replace="replace"** attribute to a configuration element produces the same behavior as a **replace:** tag in a **load replace** operation.

By default, Junos OS merges the incoming configuration and the candidate configuration. New statements and hierarchies are added, and conflicting statements are overridden. When generating a persistent or transient change, if you add the **replace="replace"** attribute to a configuration element, Junos OS replaces the existing configuration element with the incoming configuration element. If the **replace="replace"** attribute is added to a configuration element, but there is no existing element of the same name in the current configuration, the incoming configuration element is added into the configuration. Elements that do not have the **replace** attribute are merged into the configuration.

Persistent and transient changes are loaded before the standard Junos validation checks are performed. This means any configuration changes introduced by a commit script are validated for correct syntax. If the syntax is correct, the new configuration becomes the active, operational device configuration.

Protected elements in the configuration hierarchy cannot be modified or deleted by either a persistent or a transient change. If a commit script attempts to modify or delete a protected statement or hierarchy, Junos OS issues a warning that the change cannot be made, and proceeds with the commit.

Persistent and transient changes have several important differences, as described in Table 1 on page 14.

### Table 1: Differences Between Persistent and Transient Changes

| Persistent Changes | Transient Changes |
|---|---|
| A persistent change is represented in a commit script by the **<change>** tag.<br><br>Another way to represent a persistent change is with the **content** parameter inside a call to the **jcs:emit-change** template.<br><br>The **jcs:emit-change** template is a helper template contained in the **junos.xsl** import file. | A transient change is represented in a commit script by the **<transient-change>** tag.<br><br>Another way to represent a transient change is to use the **content** parameter and the **tag transient** parameter inside a call to the **jcs:emit-change** template. |
| You can use persistent changes to perform any Junos XML protocol operation, such as activate, deactivate, delete, insert (reorder), comment (annotate), and replace sections of the configuration. | Like persistent changes, you can use transient changes to perform any Junos XML protocol operation. However, some Junos XML protocol operations do not make sense to use with transient changes, such as generating comments and inactive settings. |

Table 1: Differences Between Persistent and Transient Changes *(continued)*

| Persistent Changes | Transient Changes |
|---|---|
| Persistent changes are always loaded during the commit process if no errors are generated by any commit scripts or by the standard Junos OS validity check. | For transient changes to be loaded, you must include the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level. If you enable a commit script that generates transient changes and you do not include the **allow-transients** statement in the configuration, the CLI generates an error message and the commit operation fails.<br><br>Like persistent changes, transient changes must pass the standard Junos OS validity check.<br><br>You cannot use a commit script to generate the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level. Rather, you must include this statement directly by using the CLI. |
| Persistent changes work like the **load replace** configuration mode command, and the change is added to the candidate configuration.<br><br>When generating a persistent change, if you add the **replace="replace"** attribute to a configuration element, Junos OS replaces the existing element in the candidate configuration with the incoming configuration element. If there is no existing element of the same name in the candidate configuration, the incoming configuration element is added into the configuration. Elements that do not have the **replace** attribute are merged into the configuration. | Transient changes work like the **load replace** configuration mode command, and the change is added to the checkout configuration.<br><br>When generating a transient change, if you add the **replace="replace"** attribute to a configuration element, Junos OS replaces the existing element in the checkout configuration with the incoming configuration element. If there is no existing element of the same name in the checkout configuration, the incoming configuration element is added into the configuration. Elements that do not have the **replace** attribute are merged into the configuration.<br><br>Transient changes are not copied to the candidate configuration. For this reason, transient changes are not saved in the configuration if the associated commit script is deleted or deactivated. |
| After a persistent change is committed, the software treats it like a change you make by directly editing and committing the candidate configuration.<br><br>After the persistent changes are copied to the candidate configuration, they are copied to the checkout configuration. If the changes pass the standard Junos OS validity checks, the changes are propagated to the switch, router, or security device components. | Each time a transient change is committed, the software updates the checkout configuration database. After the transient changes pass the standard Junos OS validity checks, the changes are propagated to the device components. |
| After committing a script that causes a persistent change to be generated, you can view the persistent change by issuing the **show** configuration mode command:<br><br>user@host# **show**<br><br>This command displays persistent changes only, not transient changes. | After committing a script that causes a transient change to be generated, you can view the transient change by issuing the **show \| display commit-scripts** configuration mode command:<br><br>user@host# **show \| display commit-scripts**<br><br>This command displays both persistent and transient changes. |

Table 1: Differences Between Persistent and Transient Changes *(continued)*

| Persistent Changes | Transient Changes |
|---|---|
| Persistent changes must conform to your custom configuration design rules as dictated by commit scripts.<br><br>This does not become apparent until after a second commit operation because persistent changes are not evaluated by commit script rules on the current commit operation. The subsequent commit operation fails if the persistent changes do not conform to the rules imposed by the commit scripts configured during the first commit operation. | Transient changes are never tested by and do not need to conform to your custom rules. This is caused by the order of operations in the Junos OS commit model, which is explained in detail in "Commit Scripts and the Junos OS Commit Model" on page 8. |
| A persistent change remains in the configuration even if you delete, disable, or deactivate the commit script instructions that generated the change. | If you delete, disable, or deactivate the commit script instructions that generate a transient change, the change is removed from the configuration after the next commit operation. In short, if the associated instructions or the entire commit script is removed, the transient change is also removed. |
| As with direct CLI configuration, you can remove a persistent change by rolling back to a previous configuration that did not include the change and issuing the **commit** command. However, if you do not disable or deactivate the associated commit script, and the problem that originally caused the change to be generated still exists, the change is automatically regenerated when you issue another **commit** command. | You cannot remove a transient change by rolling back to a previous configuration. |
| You can alter persistent changes directly by editing the configuration using the CLI. | You cannot directly alter or delete a transient change by using the Junos OS CLI, because the change is not in the candidate configuration.<br><br>To alter the contents of a transient change, you must alter the statements in the commit script that generates the transient change. |

## Interaction of Configuration Changes and Configuration Groups

Any configuration change you can make by directly editing the configuration using the Junos OS command-line interface (CLI) can also be generated by a commit script as a persistent or transient change. This includes values specified at a specific hierarchy level or in configuration groups. As with direct CLI configuration, values specified in the *target* override values inherited from a configuration group. The target is the statement to which you apply a configuration group by including the **apply-groups** statement.

If you define persistent or transient changes as belonging to a configuration group, the configuration groups are applied in the order you specify in the **apply-groups** statements, which you can include at any hierarchy level except the top level. You can also disable inheritance of a configuration group by including the **apply-groups-except** statement at any hierarchy level except the top level.

CAUTION: Each commit script inspects the postinheritance view of the configuration. If a candidate configuration contains a configuration group,

be careful when using a commit script to change the related target configuration, because doing so might alter the intended inheritance from the configuration group.

Also be careful when using a commit script to change a configuration group, because the configuration group might be generated by an application that performs a **load replace** operation on the group during each commit operation.

For more information about configuration groups, see the *Junos OS CLI User Guide*.

## Tag Elements and Templates for Generating Changes

To generate changes, you can use the **jcs:emit-change** template, which implicitly includes **<change>** and **<transient-change>** XML elements; or you can explicitly include **<change>** and **<transient-change>** XML elements. Using the **jcs:emit-change** template allows you to set the hierarchical context of the change once rather than multiple times.

The **<change>** and **<transient-change>** elements are similar to the **<load-configuration>** operation defined by the Junos XML management protocol. The possible contents of the **<change>** and **<transient-change>** elements are the same as the contents of the **<configuration>** tag element used in the Junos XML protocol operation **<load-configuration>**. For complete details about the **<load-configuration>** element, see the *Junos XML Management Protocol Guide*.

CHAPTER 4

# Creating Custom Configuration Syntax with Macros



## Overview of Creating Custom Configuration Syntax with Macros

Using commit script macros, you can create a custom configuration language based on simplified syntax that is relevant to your network design. This means you can use your own aliases for frequently used configuration statements.

Commit scripts generally impose restrictions on the Junos OS configuration and automatically correct configuration mistakes when they occur (as discussed in "Overview of Generating Persistent or Transient Configuration Changes" on page 13). However, macros are useful for an entirely different reason. Commit scripts that contain macros do not generally correct configuration mistakes, nor do they necessarily restrict configuration. Instead, they provide a way to simplify and speed configuration tasks, thereby preventing mistakes from occurring at all.

For a detailed example of how macros can save time and effort, see "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 97.

**Related Documentation**

- How Macros Work on page 19
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 75
- Example: Creating Custom Configuration Syntax with Macros on page 77

## How Macros Work

Your custom syntax serves as input to a commit script. The output of the commit script is standard Junos OS configuration syntax, as shown in Figure 4 on page 20. The standard Junos OS statements are added to the configuration to cause your intended operational changes.

Figure 4: Macro Input and Output


Macros use either permanent or transient change elements to expand your custom syntax into standard Junos OS configuration statements. If you use transient changes, the custom syntax appears in the candidate configuration, and the standard Junos OS syntax is copied to the checkout configuration only. If you use persistent changes, both the custom syntax and the standard Junos OS syntax appear in the candidate configuration.

This section discusses the following topics:



## Creating a Custom Syntax

Macros work by locating **apply-macro** statements that you include in the candidate configuration and using the values specified in the **apply-macro** statement as parameters to a set of instructions defined in a commit script. In effect, your custom configuration syntax serves a dual purpose. The syntax allows you to simplify your configuration tasks, and it provides to the script the data necessary to generate a complex configuration.

To enter custom syntax, you include the **apply-macro** statement at any hierarchy level and specify any data that you want inside the **apply-macro** statement:

```
apply-macro macro-name {
  parameter-name parameter-value;
}
```

You can include the **apply-macro** statement at any level of the configuration hierarchy. In this sense, the **apply-macro** statement is similar to the **apply-groups** statement. Each **apply-macro** statement must be uniquely named, relative to other **apply-macro** statements at the same hierarchy level.

An **apply-macro** statement can contain a set of parameters with optional values. The corresponding commit script can refer to the macro name, its parameters, or the parameters' values. When the script inspects the configuration and finds the data, the script performs the actions specified by a persistent or transient change element.

For example, given the following configuration stanza, you can write script instructions to generate a standard configuration based on the name of the parameter:

```
protocols {
  mpls {
    apply-macro blue-type-lsp {
      color blue;
    }
  }
}
```

The following **<xsl:for-each>** programming instruction finds **apply-macro** statements at the **[edit protocols mpls]** hierarchy level that contain a parameter named **color**:

```
<xsl:for-each select="protocols/mpls/apply-macro[data/name = 'color']">
```

The following instruction creates a variable named **color** and assigns to the variable the value of the **color** parameter, which in this case is **blue**:

```
<xsl:variable name="color" select="data[name = 'color']/value"/>
```

The following instruction adds the **admin-groups** statement to the configuration and assigns the value of the **$color** variable to the group name:

```
<transient-change>
  <protocols>
    <mpls>
      <admin-groups>
        <name>
          <xsl:value-of select="$color"/>
        </name>
      </admin-groups>
    </mpls>
  </protocols>
</transient-change>
```

The resulting configuration statements are as follows:

```
protocols {
  mpls {
    admin-groups {
      blue;
    }
  }
}
```

## <data> Element

In the XML rendering of the custom syntax within an **apply-macro** statement, parameters and their values are contained in **<name>** and **<value>** elements, respectively. The **<name>** and **<value>** elements are sibling children of the **<data>** element. For example, the **apply-macro blue-type-lsp** statement contains six parameters, as follows:

```
[edit protocols mpls]
apply-macro blue-type-lsp {
  10.1.1.1;
  10.2.2.2;
  10.3.3.3;
  10.4.4.4;
  color blue;
  group-value 0;
}
```

The parameters and values are rendered in Junos XML tag elements as follows:

```
[edit protocols mpls]
user@host# show | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
  <configuration>
    <protocols>
```

```
        <mpls>
          <apply-macro>
            <name>blue-type-lsp</name>
            <data>
              <name>10.1.1.1</name>
            </data>
            <data>
              <name>10.2.2.2</name>
            </data>
            <data>
              <name>10.3.3.3</name>
            </data>
            <data>
              <name>10.4.4.4</name>
            </data>
            <data>
              <name>color</name>
              <value>blue</value>
            </data>
            <data>
              <name>group-value</name>
              <value>0</value>
            </data>
          </apply-macro>
        </mpls>
      </protocols>
    </configuration>
  </rpc-reply>
```

When you write commit script macros, referring to the **<data>**, **<name>**, and **<value>** elements enables you to extract and manipulate the parameters contained in **apply-macro** statements. For example, in the following **select** attribute, the XPath expression extracts the text contained in the **<value>** element that is a child of a **<data>** element that also contains a **<name>** child element with the text **color**. The variable declaration assigns the text of the **<value>** element to a variable named **$color**.

```
<xsl:variable name="color" select="data[name = 'color']/value"/>
```

## Expanding the Custom Syntax

In the corresponding commit script, you include one or more XSLT or SLAX programming instructions that inspect the configuration for the **apply-macro** statement at a specified hierarchy level. Optionally, you can use the **data/name** expression to select a parameter in the **apply-macro** statement:

```
<xsl:for-each select="xpath-expression/apply-macro[data/name = 'parameter-name']">
```

For example, the following XSLT programming instruction selects every **apply-macro** statement that contains the **color** parameter and that appears at the **[edit protocols mpls]** hierarchy level:

```
<xsl:for-each select="protocols/mpls/apply-macro[data/name = 'color']">
```

The SLAX equivalent is:

```
for-each (protocols/mpls/apply-macro[data/name = 'color'])
```

When expanding macros, a particularly useful programming instruction is the **<xsl:value-of>** instruction. This instruction selects a parameter value and uses it to build option values for Junos OS statements. For example, the following instruction concatenates the value of the **$color** variable, the text **-lsp-**, and the current context node (represented by "**.**" ) to build a name for an LSP.

```
<label-switched-path>
  <name>
    <xsl:value-of select="concat($color, '-lsp-', .)"/>
  </name>
</label-switched-path>
```

SLAX uses the underscore (_) to concatenate values:

```
<label-switched-path> {
  <name> $color _ '-lsp-' _ .;
```

When the script includes instructions to find the necessary data, you can provide content for a transient change that uses the data to construct a standard Junos OS configuration.

The following transient change creates an administration group and adds the **label-switched-path** statement to the configuration. The label-switched path is assigned a name that concatenates the value of the **$color** variable, the text **-lsp-**, and the currently selected IP address represented by the period ("**.**"). The transient change also adds the **to** statement and assigns the currently selected IP address. Finally, the transient change adds the **admin-group include-any** statement and assigns the value of the **$color** variable.

```
<transient-change>
  <protocols>
    <mpls>
      <admin-groups>
        <name><xsl:value-of select="$color"/></name>
        <group-value><xsl:value-of select="$group-value"/></group-value>
      </admin-groups>
      <xsl:for-each select="data[not(value)]/name">
        <label-switched-path>
          <name><xsl:value-of select="concat($color, '-lsp-', .)"/></name>
          <to><xsl:value-of select="."/></to>
          <admin-group>
            <include-any><xsl:value-of select="$color"/></include-any>
          </admin-group>
        </label-switched-path>
      </xsl:for-each>
    </mpls>
  </protocols>
</transient-change>
```

The SLAX equivalent is:

```
<transient-change> {
  <protocols> {
    <mpls> {
      <admin-groups> {
        <name> $color;
        <group-value> $group-value;
      }
      for-each (data[not(value)]/name) {
```

```
            <label-switched-path> {
              <name> $color _ '-lsp-' _ .;
              <to> .;
              <admin-group> {
                <include-any> $color;
              }
            }
          }
        }
      }
    }
```

NOTE: The example shown here is partial. For a full example, see "Example: Creating Custom Configuration Syntax with Macros" on page 77.

After committing the configuration, the script runs, and the resulting full configuration looks like this:

```
[edit]
protocols {
  mpls {
    label-switched-path blue-lsp-10.1.1.1 {
      to 10.1.1.1;
      admin-group include-any blue;
    }
    label-switched-path blue-lsp-10.2.2.2 {
      to 10.2.2.2;
      admin-group include-any blue;
    }
    label-switched-path blue-lsp-10.3.3.3 {
      to 10.3.3.3;
      admin-group include-any blue;
    }
    label-switched-path blue-lsp-10.4.4.4 {
      to 10.4.4.4;
      admin-group include-any blue;
    }
  }
}
```

The previous example demonstrates how you can use a simplified custom syntax to configure label-switched paths (LSPs). If your network design requires a large number of LSPs to be configured, using a commit script macro can save time, ensure consistency, and prevent configuration errors.

## Other Ways to Use Macros

The example discussed in "Creating a Custom Syntax" on page 20 shows a macro that uses transient changes to create the intended operational impact. Alternatively, you can create a commit script that uses persistent changes to add the standard Junos OS statements to the candidate configuration and delete your custom syntax entirely. This way, a network operator who might be unfamiliar with your custom syntax can view the configuration file and see the full configuration rendered as standard Junos OS statements.

Still, because the commit script macro remains in effect, you can quickly and easily create a complex configuration using your custom syntax.

In addition to the type of application discussed in "Creating a Custom Syntax" on page 20, you can also use macros to prevent a commit script from performing a task. For example, a basic commit script that automatically adds MPLS configuration to interfaces can make an exception for interfaces you explicitly tag as not requiring MPLS, by testing for the presence of an **apply-macro** statement named **no-mpls**. For an example of this use of macros, see "Example: Controlling LDP Configuration" on page 126.

You can use the **apply-macro** statement as a place to store external data. The commit script does not inspect the **apply-macro** statement, so the **apply-macro** statement has no operational impact on the device, but the data can be carried in the configuration file to be used by external applications.

**Related Documentation**

- Overview of Creating Custom Configuration Syntax with Macros on page 19
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 75
- Example: Creating Custom Configuration Syntax with Macros on page 77

PART 2

# Configuration

CHAPTER 5

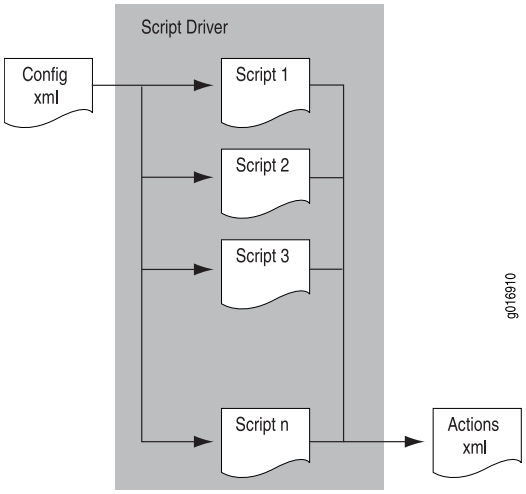
# Creating and Executing Commit Scripts



## Avoiding Potential Conflicts When Using Multiple Commit Scripts

When you use multiple commit scripts, each script evaluates the original candidate configuration file. Changes made by one script are not evaluated by the other scripts. This means that conflicts between scripts might not be resolved when the scripts are first applied to the configuration. The commit scripts are executed in the order they are listed at the **[edit system scripts commit]** hierarchy level, as illustrated in Figure 5 on page 29.

### Figure 5: Configuration Evaluation by Multiple Commit Scripts



As an example of a conflict between commit scripts, suppose that commit script **A.xsl** is created to ensure that the device uses the domain name (DNS) server with IP address **192.168.0.255**. Later, the DNS server's address is changed to **192.168.255.255** and a second script, **B.xsl**, is added to check that the device uses the DNS server with that address. However, script **A.xsl** is not removed or disabled.

Because each commit script evaluates the original candidate configuration, the final result of executing both scripts **A.xsl** and **B.xsl** depends on which DNS server address is configured in the original candidate configuration. If the now outdated address of **192.168.0.255** is configured, script **B.xsl** changes it to **192.168.255.255**. However, if the correct address of **192.168.255.255** is configured, script **A.xsl** changes it to the incorrect value **192.168.0.255**.

As another example of a potential conflict between commit scripts, suppose that a commit script protects a hierarchy using the **protect** attribute. If a second commit script attempts to modify or delete the hierarchy or the statements within the hierarchy, Junos OS issues a warning during the commit process and prevents the configuration change.

Exercise care to ensure that you do not introduce conflicts between scripts like those described in the examples. As a method of checking for conflicts with persistent changes, you can issue two separate **commit** commands.

**Related Documentation**

- How Commit Scripts Work on page 5

## Required Boilerplate for Commit Scripts

When you write commit scripts, you use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided with Junos OS. These tools include basic boilerplate that you must include in all commit scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make commit scripts easier to read and write, which you import from a file called **junos.xsl**. For more information about the extension functions and templates, see Junos Script Automation: Extension Functions in the jcs Namespace Overview and Junos Script Automation: Named Templates in the jcs Namespace Overview.

Commit scripts are based on Junos XML and Junos XML protocol tag elements. Like all XML elements, angle brackets enclose the name of a Junos XML or Junos XML protocol tag element in its opening and closing tags. This is an XML convention, and the brackets are a required part of the complete tag element name. They are not to be confused with the angle brackets used in the documentation to indicate optional parts of Junos OS CLI command strings.

You must include either XSLT or SLAX boilerplate as the starting point for all commit scripts that you create. The XSLT boilerplate follows:

**XSLT Boilerplate for Commit Scripts**

```
1 <?xml version="1.0" standalone="yes"?>
2 <xsl:stylesheet version="1.0"
3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4    xmlns:junos="http://xml.juniper.net/junos/*/junos"
5    xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6    xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
```

```
 7    <xsl:import href="../import/junos.xsl"/>

 8    <xsl:template match="configuration">
       <!-- ... Insert your code here ... -->
 9    </xsl:template>
10  </xsl:stylesheet>
```

Line 1 is the Extensible Markup Language (XML) processing instruction (PI). This PI specifies that the code is written in XML using version 1.0. The XML PI, if present, must be the first noncomment token in the script file.

```
1  <?xml version="1.0"?>
```

Lines 2 through 6 set the style sheet element and the associated namespaces. Line 2 sets the style sheet version as 1.0. Lines 3 through 6 list all the namespace mappings commonly used in commit scripts. Not all of these prefixes are used in this example, but it is not an error to list namespace mappings that are not referenced. Listing all namespace mappings prevents errors if the mappings are used in later versions of the script.

```
2  <xsl:stylesheet version="1.0"
3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4    xmlns:junos="http://xml.juniper.net/junos/*/junos"
5    xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6    xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
```

Line 7 is an XSLT import statement. It loads the templates and variables from the file referenced as **../import/junos.xsl**, which ships as part of Junos OS. The **junos.xsl** file contains a set of named templates you can call in your scripts. These named templates are discussed in Junos Script Automation: Named Templates in the jcs Namespace Overview and Junos Named Templates in the jcs Namespace Summary.

```
7    <xsl:import href="../import/junos.xsl"/>
```

Line 8 defines a template that matches the **<configuration>** element, which is the node selected by the **<xsl:template match="/">** template, contained in the **junos.xsl** import file. The **<xsl:template match="configuration">** element allows you to exclude the **/configuration/** root element from all XML Path Language (XPath) expressions in the script and begin XPath expressions with the top Junos OS hierarchy level. For more information, see XPath Overview.

```
8    <xsl:template match="configuration">
```

Add your code between Lines 8 and 9.

Line 9 closes the template.

```
9    </xsl:template>
```

Line 10 closes the style sheet and the commit script.

```
10  </xsl:stylesheet>
```

### SLAX Boilerplate for Commit Scripts

The corresponding SLAX boilerplate is as follows:

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
```

```
match configuration {
  /*
  * Insert your code here
  */
}
```

## XML Syntax for Common Commit Script Tasks

A commit script can perform common configuration tasks by adding the appropriate attribute to a specific XML tag. Table 2 on page 32 summarizes the tasks and the syntax for each task.

### Table 2: XML Syntax for Common Commit Script Tasks

| Action | Syntax | Example |
|---|---|---|
| Add a data element | normal XML | <address><br><name>192.168.1.1</name><br></address> |
| Remove the inactive tag from a statement | active="active" | <address active="active"><br><name>192.168.1.1/30</name><br></address> |
| Delete a data element | delete="delete" | <address delete="delete"><br><name>192.168.1.1/30</name><br></address> |
| Add the inactive tag to a statement | inactive="inactive" | <address inactive="inactive"><br><name>192.168.1.1/30</name><br></address> |
| Insert a new ordered data element | insert="(before \| after)" name="*reference-value*" | <address insert="before" name="192.168.1.5/30"><br><name>192.168.1.1/30</name><br></address> |
| Add the protect tag to a statement or node to prevent configuration changes to that element | protect="protect" | <address protect="protect""><br><name>192.168.1.1/30</name><br></address> |
| Rename a statement | rename="rename" name="*new-name*" | <address rename="rename" name="192.168.1.1/30"><br><name>192.168.1.5/30</name><br></address> |
| Replace a node or statement in the hierarchy | replace="replace" | <system><br><services replace="replace"><br>[...]<br></services><br></system> |

Table 2: XML Syntax for Common Commit Script Tasks *(continued)*

| Action | Syntax | Example |
|---|---|---|
| Unprotect a statement or node in the hierarchy | unprotect="unprotect" | <address unprotect="unprotect"><br><name>192.168.1.1/30</name><br></address> |
| Annotate a configuration statement with a comment | <junos:comment> | <system><br><junos:comment><br>/* added by *username* */<br></junos:comment><br><services><br>[...]<br></services><br></system> |

## Design Considerations for Commit Scripts

After you have an understanding of XSLT and some experience looking at Junos OS configuration data in XML, creating commit scripts is fairly straightforward. This section provides some advice and common patterns for developing commit scripts.

XSLT is an interpreted language, making performance an important consideration. For best performance, minimize node traversals and testing performed on each node. When possible, use the **select** attribute on a recursive **<xsl:apply-templates>** invocation to limit the portion of the document hierarchy being visited.

For example, the following **select** attribute limits the nodes to be evaluated by specifying SONET/SDH interfaces that have the **inet** (IPv4) protocol family enabled:

```
<xsl:apply-templates select="interfaces/interface[starts-with(name, 'so-') and
unit/family/inet]"/>
```

The following example contains two **<xsl:apply-templates>** instructions that limit the scope of the script to the **import** statements configured at the **[edit protocols ospf]** and **[edit protocols isis]** hierarchy levels:

```
<xsl:template match="configuration">
  <xsl:apply-templates select="protocols/ospf/import"/>
  <xsl:apply-templates select="protocols/isis/import"/>
  <!-- ... body of template ... -->
</xsl:template>
```

In an interpreted language, doing anything more than once can affect performance. If the script needs to reference a node or node set repeatedly, make a variable that holds the node set, and then make multiple references to the variable. For example, the following variable declaration creates a variable called **mpls** that resolves to the **[edit protocols mpls]** hierarchy level. This allows the script to traverse the **/protocols/** hierarchy searching for the **mpls/** node only once.

```
<xsl:variable name="mpls" select="/protocols/mpls"/>
<xsl:choose>
  <xsl:when test="$mpls/path-mtu/allow-fragmentation">
    <!-- ... -->
  </xsl:when>
```

```
    <xsl:when test="$mpls/hop-limit &gt; 40">
      <!-- ... -->
    </xsl:when>
  </xsl:choose>
```

Variables are also important when using **<xsl:for-each>** instructions, because the current context node examines each node selected by the **<xsl:for-each>** instruction. For example, the following script uses multiple variables to store and refer to values as the **<xsl:for-each>** instruction evaluates the E1 interfaces that are configured on all channelized STM1 (**cstm1-**) interfaces:

```
<xsl:param name="limit" select="16"/>
<xsl:template match="configuration">
  <xsl:variable name="interfaces" select="interfaces"/>
  <xsl:for-each select="$interfaces/interface[starts-with(name, 'cstm1-')]">
    <xsl:variable name="triple" select="substring-after(name, 'cstm1-')"/>
    <xsl:variable name="e1name" select="concat('e1-', $triple)"/>
    <xsl:variable name="count"
          select="count($interfaces/interface[starts-with(name, $e1name)])"/>
    <xsl:if test="$count > $limit">
      <xnm:error>
        <edit-path>[edit interfaces]</edit-path>
        <statement><xsl:value-of select="name"/></statement>
        <message>
          <xsl:text>E1 interface limit exceeded on CSTM1 IQ PIC. </xsl:text>
          <xsl:value-of select="$count"/>
          <xsl:text> E1 interfaces are configured, but only </xsl:text>
          <xsl:value-of select="$limit"/>
          <xsl:text> are allowed.</xsl:text>
        </message>
      </xnm:error>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

If you channelize a **cstm1-0/1/0** interface into 17 **E1** interfaces, the script causes the following error message to appear when you issue the **commit** command. (For more information about this example, see "Example: Limiting the Number of E1 Interfaces" on page 143.)

```
[edit]

user@host# commit
[edit interfaces]
    'cstm1-0/1/0'
    E1 interface limit exceeded on CSTM1 IQ PIC.
    17 E1 interfaces are configured, but only 16 are allowed.
error: 1 error reported by commit scripts
error: commit script failure
```

**Related Documentation**

-

## Line-by-Line Explanation of Sample Commit Scripts



### Applying a Change to SONET/SDH Interfaces

The following commit script applies a transient change to each interface whose name begins with **so-**, setting the encapsulation to **ppp**. For information about transient changes, see "Overview of Generating Persistent or Transient Configuration Changes" on page 13. For a SLAX version of this example, see "Example: Generating a Transient Change" on page 69.

```
 1  <?xml version="1.0"?>
 2  <xsl:stylesheet version="1.0"
 3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 4    xmlns:junos="http://xml.juniper.net/junos/*/junos"
 5    xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
 6    xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
 7    <xsl:import href="../import/junos.xsl"/>

 8    <xsl:template match="configuration">
 9      <xsl:for-each select="interfaces/interface[starts-with(name, 'so-') \
         and unit/family/inet]">
10        <transient-change>
11          <interfaces>
12            <interface>
13              <name><xsl:value-of select="name"/></name>
14              <encapsulation>ppp</encapsulation>
15            </interface>
16          </interfaces>
17        </transient-change>
18      </xsl:for-each>
19    </xsl:template>
20  </xsl:stylesheet>
```

Lines 1 through 8 are boilerplate as described in "Required Boilerplate for Commit Scripts" on page 30 and are omitted here for brevity.

Line 9 is an **<xsl:for-each>** programming instruction that examines each interface node whose names starts with **so-** and that has **family inet** enabled on any logical unit. (It appears here on two lines only for brevity.)

```
 9      <xsl:for-each select="interfaces/interface[starts-with(name, 'so-') \
         and unit/family/inet]">
```

Line 10 is the open tag for a transient change. The possible contents of the **<transient-change>** element are the same as the contents of the **<configuration>** tag element in the Junos XML protocol operation **<load-configuration>**.

```
10        <transient-change>
```

Lines 11 through 16 represent the content of the transient change. The encapsulation is set to **ppp**.

```
11          <interfaces>
```

```
12              <interface>
13                <name><xsl:value-of select="name"/></name>
14                <encapsulation>ppp</encapsulation>
15              </interface>
16            </interfaces>
```

Lines 17 through 19 close all open tags in this template.

```
17          </transient-change>
18        </xsl:for-each>
19      </xsl:template>
```

Line 20 closes the style sheet and the commit script.

```
20  </xsl:stylesheet>
```

## Applying a Change to ISO-Enabled Interfaces

The following sample script ensures that interfaces that are enabled for an International Organization for Standardization (ISO) protocol also have MPLS enabled and are included at the **[edit protocols mpls interface]** hierarchy level. For a SLAX version of this example, see "Example: Controlling IS-IS and MPLS Interfaces" on page 122.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4    xmlns:junos="http://xml.juniper.net/junos/*/junos"
5    xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6    xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
7    <xsl:import href="../import/junos.xsl"/>

8      <xsl:template match="configuration">
9        <xsl:variable name="mpls" select="protocols/mpls"/>
10       <xsl:for-each select="interfaces/interface/unit[family/iso]">
11       <xsl:variable name="ifname" select="concat(../name, '.', name)"/>
12        <xsl:if test="not(family/mpls)">
13          <xsl:call-template name="jcs:emit-change">
14            <xsl:with-param name="message">
15              <xsl:text>
16                Adding 'family mpls' to ISO-enabled interface
17              </xsl:text>
18            </xsl:with-param>
19            <xsl:with-param name="content">
20              <family>
21                <mpls/>
22              </family>
23            </xsl:with-param>
24          </xsl:call-template>
25        </xsl:if>
26        <xsl:if test="$mpls and not($mpls/interface[name = $ifname])">
27          <xsl:call-template name="jcs:emit-change">
28            <xsl:with-param name="message">
29              <xsl:text>Adding ISO-enabled interface </xsl:text>
30              <xsl:value-of select="$ifname"/>
31              <xsl:text> to [protocols mpls]</xsl:text>
32            </xsl:with-param>
33            <xsl:with-param name="dot" select="$mpls"/>
```

```
34              <xsl:with-param name="content">
35                <interface>
36                  <name>
37                    <xsl:value-of select="$ifname"/>
38                  </name>
39                </interface>
40              </xsl:with-param>
41            </xsl:call-template>
42          </xsl:if>
43        </xsl:for-each>
44      </xsl:template>
45  </xsl:stylesheet>
```

Lines 1 through 8 are boilerplate as described in "Required Boilerplate for Commit Scripts" on page 30 and are omitted here for brevity.

Line 9 saves a reference to the **[edit protocols mpls]** hierarchy level so that it can be referenced in the following **for-each** loop.

```
9           <xsl:variable name="mpls" select="protocols/mpls"/>
```

Line 10 examines each interface unit (logical interface) on which ISO is enabled. The **select** stops at the **unit**, but the predicate limits the selection to only those units that contain an **<iso>** element nested under a **<family>** element.

```
10          <xsl:for-each select="interfaces/interface/unit[family/iso]">
```

Line 11 builds the interface name in a variable. First, the **name** attribute of the variable declaration is set to **ifname**. In Junos OS, an interface name is the concatenation of the device name, a period, and the unit number. At this point in the script, the context node is the unit number, because Line 10 changes the context to interfaces/interface/unit. The **../name** refers to the **<name>** element of the parent node of the context node, which is the device name (***type-fpc/pic/port***). The "**name**" token in the XPath expression refers to the **<name>** element of the context node, which is the unit number (***unit-number***). After the concatenation is performed, the XPath expression in Line 11 resolves to ***type-fpc/pic/port.unit-number***. As the **<xsl:for-each>** instruction in Line 10 traverses the hierarchy and locates ISO-enabled interfaces, the interface names are recursively stored in the **ifname** variable.

```
11          <xsl:variable name="ifname" select="concat(../name, '.', name)"/>
```

Line 12 evaluates as true for each ISO-enabled interface that does not have MPLS enabled.

```
12          <xsl:if test="not(family/mpls)">
```

Line 13 calls the **jcs:emit-change** template, which is a helper or convenience template in the **junos.xsl** file. This template is discussed in jcs:emit-change Template.

```
13            <xsl:call-template name="jcs:emit-change">
```

Lines 14 through 18 use the **message** parameter from the **jcs:emit-change** template. The message parameter is a shortcut you can use instead of explicitly including the **<warning>**, **<edit-path>**, and **<statement>** elements.

```
14              <xsl:with-param name="message">
15                <xsl:text>
16                  Adding 'family mpls' to ISO-enabled interface
17                </xsl:text>
18              </xsl:with-param>
```

Lines 19 through 23 use the **content** parameter from the **jcs:emit-change** template. The **content** parameter specifies the change to make, relative to the current context node.

```
19          <xsl:with-param name="content">
20            <family>
21              <mpls/>
22            </family>
23          </xsl:with-param>
```

Lines 24 and 25 close the tags opened in Lines 13 and 12, respectively.

```
24          </xsl:call-template>
25        </xsl:if>
```

Line 26 tests whether MPLS is already enabled and if this interface is not configured at the **[edit protocols mpls interface]** hierarchy level.

```
26        <xsl:if test="$mpls and not($mpls/interface[name = $ifname])">
```

Lines 27 through 41 contain another invocation of the **jcs:emit-change** template. In this invocation, the interface is added at the **[edit protocols mpls interface]** hierarchy level.

```
27            <xsl:call-template name="jcs:emit-change">
28              <xsl:with-param name="message">
29                <xsl:text>Adding ISO-enabled interface </xsl:text>
30                <xsl:value-of select="$ifname"/>
31                <xsl:text> to [edit protocols mpls]</xsl:text>
32              </xsl:with-param>
33              <xsl:with-param name="dot" select="$mpls"/>
34              <xsl:with-param name="content">
35                <interface>
36                <name>
37                  <xsl:value-of select="$ifname"/>
38                </name>
39              </interface>
40            </xsl:with-param>
41          </xsl:call-template>
```

Lines 42 through 45 close all open elements.

```
42        </xsl:if>
43      </xsl:for-each>
44    </xsl:template>
45  </xsl:stylesheet>
```

**Related Documentation**

- Example: Generating a Transient Change on page 69

## Controlling Execution of Commit Scripts During Commit Operations

Commit scripts are stored on a device's hard drive in the **/var/db/scripts/commit** directory or on the flash drive in the **/config/scripts/commit** directory. Only users in the Junos OS superuser login class can access and edit files in these directories. For information about setting the storage location for scripts, see Storing and Enabling Scripts and Storing Scripts in Flash Memory. A commit script is not actually executed during commit operations unless its filename is included at the **[edit system scripts commit file]** hierarchy

level. To prevent execution of a commit script, delete the commit script's filename at that hierarchy level.

By default, the commit operation fails unless all scripts included at the **[edit system scripts commit file]** hierarchy level actually exist in the commit script directory. To enable the commit operation to succeed even if a script is missing, include the **optional** statement at the **[edit system scripts commit file *filename*]** hierarchy level. For example, you might want to mark a script as optional if you anticipate the need to quickly remove it from operation by deleting it from the commit script directory, but do not want to remove the commit script filename at the **[edit system scripts commit file]** hierarchy level. To enable use of the script again later, you simply replace the file in the commit script directory.

CAUTION: When you include the **optional** statement at the **[edit system scripts commit file *filename*]** hierarchy level, no error message is generated during the commit operation if the file does not exist. As a result, you might not be aware that a script is not executed as you expect.

You can also deactivate and reactivate commit scripts by issuing the **deactivate** and **activate** configuration mode commands. When a commit script is deactivated, the script is marked as inactive in the configuration and does not execute during the commit operation. When a commit script is reactivated, the script is again executed during the commit operation.

To determine which commit scripts are currently enabled on the device, use the **show** command to display the files included at the **[edit system scripts commit]** hierarchy level. To ensure that the enabled files are on the device, list the contents of the **/var/run/scripts/commit/** directory using the **file list /var/run/scripts/commit** operational mode command.

The filename of a commit script written in SLAX must include the **.slax** extension for the script to be executed. No filename extension is required for commit scripts written in XSLT, but we strongly recommend that you append the **.xsl** extension.

See the following sections:


- Enabling Commit Scripts to Execute During Commit Operations on page 39
- Preventing Commit Scripts from Executing During Commit Operations on page 40
- Deactivating Commit Scripts on page 40
- Activating Commit Scripts on page 41


## Enabling Commit Scripts to Execute During Commit Operations

To configure a commit script to execute during a commit operation, follow these steps:

1. Ensure that the commit script is located in the correct directory: the **/var/db/scripts/commit** directory on the hard drive or the **/config/scripts/commit** directory on the flash drive. For more information about script storage location, see Storing Scripts in Flash Memory.

2. Enable the commit script by including the **file *filename*** statement at the **[edit system scripts commit]** hierarchy level. Only users who belong to the Junos OS **super-user** login class can enable commit scripts.

   ```
   [edit system scripts commit]
   user@host# set file filename <optional>
   ```

   - ***filename***—Name of the commit script.
   - **optional**—Enable the commit operation to succeed when the script file does not exist in the script directory. If this statement is omitted, the commit operation fails if the script does not exist.

3. Commit the configuration:

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

The commit script does not execute during this commit operation, but executes automatically during each subsequent commit operation.

## Preventing Commit Scripts from Executing During Commit Operations

To prevent a commit script from executing during a commit operation, follow these steps:

1. Delete the commit script filename at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   user@host# delete file filename
   ```

   ***filename***—Name of the commit script.

2. Remove the commit script from the commit script directory. Although removing the commit script from the commit script directory is not necessary, it is always a good policy to delete unused files from the system.

3. Commit your changes:

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

## Deactivating Commit Scripts

To deactivate a commit script, follow these steps:

1. Issue the **deactivate** command:

   ```
   [edit]
   user@host# deactivate system scripts commit file filename
   ```

2. Commit your changes:

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

A deactivated commit script is marked as **inactive:** and ignored during a commit operation.

In this example, the script **mycommit.slax** is deactivated:

```
[edit]
user@host# deactivate system scripts commit file mycommit.slax
[edit]
user@host# show system scripts commit
inactive: file mycommit.slax
```

### Activating Commit Scripts

To activate an inactive commit script, follow these steps:

1. Issue the **activate** command:

   ```
   [edit]
   user@host# activate system scripts commit file filename
   ```

2. Commit your changes:

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

The commit script does not execute during this commit operation, but executes automatically during each subsequent commit operation.

## Configuring Checksum Hashes for a Commit Script

You can configure one or more checksum hashes that can be used to verify the integrity of a commit script before the script runs on the switch, router, or security device.

To configure a checksum hash:

1. Create the script.
2. Place the script in the **/var/db/scripts/commit** directory on the device.
3. Run the script through one or more hash functions to calculate hash values.

   Junos OS supports MD5, SHA-1, and SHA-256 hash functions.

   ```
   user@host> file checksum md5 /var/db/scripts/commit/script1.slax
     MD5 (/var/db/scripts/commit/script1.slax) = 3af7884eb56e2d4489c2e49b26a39a97
   user@host> file checksum sha1 /var/db/scripts/commit/script1.slax
     SHA1 (/var/db/scripts/commit/script1.slax) =
     00dc690fb08fb049577d012486c9a6dad34212c0
   user@host> file checksum sha-256 /var/db/scripts/commit/script1.slax
     SHA256 (/var/db/scripts/commit/script1.slax) =
     150bf53383769f3bfedd41fe73320777f208d4fda81230cb27b8738
   ```

4. Configure the script.

   ```
   [edit system scripts commit]
   user@host# set file script1.slax checksum md5 3af7884eb56e2d4489c2e49b26a39a97
   [edit system scripts commit]
   user@host# set file script1.slax checksum
     sha-1 00dc690fb08fb049577d012486c9a6dad34212c0
   [edit system scripts commit]
   ```

```
user@host# set file script1.slax checksum
  sha-256 150bf53383769f3bfedd41fe73320777f208d4fda81230cb27b8738
```

During the execution of the script, Junos OS recalculates the checksum value using the configured hash and verifies that the calculated value matches the configured value. If the values differ, the execution of the script fails. When you configure multiple checksum values with different hash algorithms, all the configured values must match the calculated values; otherwise, the script execution fails. The commit operation also fails.

**Related Documentation**

- Configuring Checksum Hashes for an Event Script
- Configuring Checksum Hashes for an Op Script
- file checksum md5 command in the *System Basics and Services Command Reference*
- file checksum sha-256 command in the *System Basics and Services Command Reference*
- file checksum sha1 command in the *System Basics and Services Command Reference*

## Executing Large Commit Scripts

When you use large commit scripts, the standard commit model can have trouble reading these scripts. When this occurs, you can include the **direct-access** statement at the **[edit system scripts commit]** hierarchy level. When the **direct-access** statement is included, the script driver retrieves the candidate configuration directly from the configuration database. Once the candidate configuration is retrieved, the script driver processes this configuration file against the commit scripts and returns any generated actions to the management (mgd) process.

Directly accessing the configuration data and processing non-XML converted data are processor-intensive compared to the standard commit model. You should only use this feature to handle large files, because system performance is affected.

To set the script driver to directly access the candidate configuration, include the **direct-access** statement at the **[edit system scripts commit]** hierarchy level.

```
[edit system scripts commit]
direct-access;
```

CHAPTER 6

# Generating a Custom Warning, Error, or System Log Message



## Generating a Custom Warning, Error, or System Log Message

To generate a custom warning, error, or system log message, follow these steps:

1. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from "Required Boilerplate for Commit Scripts" on page 30. It is reproduced here for convenience:

### XSLT Boilerplate

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <!-- ... insert your code here ... -->
  </xsl:template>
</xsl:stylesheet>
```

### SLAX Boilerplate

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
```

```
match configuration {
  /*
   * insert your code here
  */
}
```

2. At the position indicated by the comment "***Insert your code here***," include one or more XSLT programming instructions or their SLAX equivalents. Commonly used XSLT constructs include the following:

   - **<xsl:choose> <xsl:when> <xsl:otherwise>**—Conditional construct that causes different instructions to be processed in different circumstances. The **<xsl:choose>** instruction contains one or more **<xsl:when>** elements, each of which tests an XPath expression. If the test evaluates as true, the XSLT processor executes the instructions in the **<xsl:when>** element. The XSLT processor processes only the instructions contained in the first **<xsl:when>** element whose **test** attribute evaluates as true. If none of the **<xsl:when>** elements' **test** attributes evaluate as true, the content of the **<xsl:otherwise>** element, if there is one, is processed.

   - **<xsl:for-each select="*xpath-expression*">**—Programming instruction that tells the XSLT processor to gather together a set of nodes and process them one by one. The nodes are selected by the Extensible Markup Language (XML) Path Language (XPath) expression in the **select** attribute. Each of the nodes is then processed according to the instructions contained in the **<xsl:for-each>** instruction. Code inside an **<xsl:for-each>** instruction is evaluated recursively for each node that matches the XPath expression. The context is moved to the node during each pass.

   - **<xsl:if test="*xpath-expression*">**—Conditional construct that causes instructions to be processed if the XPath expression in the **test** attribute evaluates to **true**.

     For example, the following programming instruction evaluates as true when the **source-route** statement is not included at the **[edit chassis]** hierarchy level:

     ```
     <xsl:if test="not(chassis/source-route)">
     ```

     In SLAX, the **if** construct looks like this:

     ```
     if (not(chassis/source-route))
     ```

   For more information about how to use programming instructions, including examples and pseudocode, see XSLT Programming Instructions Overview. For information about writing scripts in SLAX instead of XSLT, see SLAX Overview.

3. Include a **<xnm:warning>**, **<xnm:error>**, or **<syslog>** element with a **<message>** child element that specifies the content of the message.

   For warning and error messages, you can include several other child elements, such as the **jcs:edit-path** and **jcs:statement** templates, which cause the warning or error message to include the relevant configuration hierarchy and statement information, as shown in the following examples.

   This **<xnm:warning>** element:

   ```
   <xnm:warning>
     <xsl:call-template name="jcs:edit-path">
       <xsl:with-param name="dot" select="chassis"/>
   ```

```
    </xsl:call-template>
    <message>IP source-route processing is not enabled.</message>
  </xnm:warning>
```

emits this output when you issue the **commit** command:

```
[edit]
user@host# commit
    [edit chassis]
        warning: IP source-route processing is not enabled.
    commit complete
```

This **<xnm:error>** element:

```
<xnm:error>
  <xsl:call-template name="jcs:edit-path"/>
  <xsl:call-template name="jcs:statement"/>
  <message>Missing a description for this T1 interface.</message>
</xnm:error>
```

emits this output when you issue the **commit** command:

```
[edit]
user@host# commit
    [edit interfaces interface t1-0/0/0]
        'interface t1-0/0/0;'
        Missing a description for this T1 interface.
    error: 1 error reported by commit scripts
    error: commit script failure
```

NOTE: If you are including a warning message in conjunction with a script-generated configuration change, you can generate the warning by including the **message** parameter with the **jcs:emit-change** template. The **message** parameter causes the **jcs:emit-change** template to call the **<xnm:warning>** template, which sends a warning notification to the CLI. (For more information, see "Overview of Generating Persistent or Transient Configuration Changes" on page 13.)

For system log messages, the only supported child element is **<message>**:

```
<syslog>
  <message>syslog-string</message>
</syslog>
```

For a description of all the XSLT tags and attributes you can include, see "Tag Elements to Use When Generating Messages" on page 46.

For SLAX versions of these constructs, see "Example: Generating a Custom Warning Message" on page 48, "Example: Generating a Custom Error Message" on page 51, and "Example: Generating a Custom System Log Message" on page 54.

4. Save the script with a meaningful name.

5. Copy the script to either the **/var/db/scripts/commit** directory on the hard drive or the **/config/scripts/commit** directory on the flash drive. For information about setting the storage location for commit scripts, see Storing Scripts in Flash Memory.

   If the device has dual Routing Engines and you want the script to take effect on both of them, you must copy the script to the **/var/db/scripts/commit** or the **/config/scripts/commit** directory on both Routing Engines. The **commit synchronize** command does not copy scripts between Routing Engines.

6. Enable the script by including the **file** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   file filename;
   ```

   where ***filename*** is the name of the script.

**Related Documentation**

- Example: Generating a Custom Error Message on page 51
- Example: Generating a Custom System Log Message on page 54
- Example: Generating a Custom Warning Message on page 48

## Tag Elements to Use When Generating Messages

Table 3 on page 46 summarizes the tag elements that you can include in a custom warning, error, or system log message. For examples, see "Example: Generating a Custom Warning Message" on page 48, "Example: Generating a Custom Error Message" on page 51, and "Example: Generating a Custom System Log Message" on page 54.

### Table 3: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|---|---|---|
| Container Tags and Attributes | | |
| **<syslog>** | Required for system log messages | Indicates that a system log message is going to be recorded. |
| **<xnm:error>** | Required for error messages | Indicates that the server has encountered a problem while processing the client application's request. |
| **<xnm:warning>** | Required for warning messages | Indicates that the server has encountered a problem while processing the client application's request. |
| **xmlns** ***url*** | Supported in warning and error messages | Names the XML namespace for the contents of the tag element. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**. |

Table 3: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages *(continued)*

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|---|---|---|
| **xmlns:xnm** *url* | Required for warning and error messages. The **xmlns:xnm** element is included in the script boilerplate, which sets the namespace globally. | Names the XML namespace for child tag elements that have the **xnm:** prefix on their names. The value is a URL of the form **http://xml.juniper.net/xnm/***version***/xnm**, where ***version*** is a string such as **1.1**. |
| Content Tags | | |
| **<column>** | Supported in warning and error messages only | Identifies the element that caused the error by specifying its position as the number of characters after the first character in the line specified by the **<line-number>** tag element in the configuration file that was being loaded (which is named in the **<filename>** tag element). We recommend combining the **<column>** tag with the **<line-number>** and **<filename>** tags. |
| **<database-status-information>** | Supported in error messages only | Provides information about the users currently editing the configuration. |
| **<edit-path>** | Supported in warning and error messages only | Specifies the level in the configuration hierarchy where the problem occurred, using the CLI configuration mode banner. We recommend combining the **<edit-path>** tag with the **<statement>** tag. |
| **<filename>** | Supported in warning and error messages only | Names the configuration file that was being loaded. |
| **<line-number>** | Supported in warning and error messages only | Specifies the line number where the error occurred in the configuration file that was being loaded, which is named by the **<filename>** tag element. We recommend combining the **<line-number>** tag with the **<column>** and **<filename>** tags. |
| **<message>** | Required in warning, error, and system log messages | Describes the warning, error, or system log message in a natural-language text string. |
| **<parse/>** | Supported in error messages only | Indicates that there was a syntactic error in the request submitted by the client application. |
| **<reason>** | Supported in warning and error messages only | Describes the reason for the warning or error message. |
| **<re-name>** | Supported in warning and error messages only | Names the Routing Engine on which the process named by the **<source-daemon>** tag element is running. |
| **<source-daemon>** | Supported in warning and error messages only | Names the Junos OS module that was processing the request in which the warning or error message occurred. |
| **<statement>** | Supported in warning and error messages only | Specifies the configuration statement in effect when the problem occurred. We recommend combining the **<statement>** tag with the **<edit-path>** tag. |

Table 3: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages *(continued)*

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|---|---|---|
| **<token>** | Supported in warning and error messages only | Names the element in the request that caused the warning or error message. |
| **<xsl:call-template name="jcs:edit-path">** | Supported in warning and error messages only | Emits an **<edit-path>** element, which specifies the CLI configuration mode edit path in effect when the warning or error was generated.<br><br>If the problem is not at the current position in the XML hierarchy, you can alter the edit path by passing the **dot** parameter. For example, **<xsl:param name="dot" select="system/ports/console"/>** changes the edit path to **[edit system ports console]**. |
| **<xsl:call-template name="jcs:statement">** | Supported in warning and error messages only | Emits a **<statement>** element, which describes the configuration statement in effect when the warning or error was generated.<br><br>If the problem is not at the current position in the XML hierarchy, you can alter the statement by passing the **dot** parameter. For example, **<xsl:with-param name="dot" select="system/ports/console/type"/>** changes the statement to **type**. |

## Example: Generating a Custom Warning Message

This example commit script generates a custom warning message when a specific statement is not included in the device configuration. The commit process is not affected by warnings.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

Using a commit script, write a custom warning message that appears when the **source-route** statement is not included at the **[edit chassis]** hierarchy level. (This example is the complete script for the sample **<xnm:warning>** element used in "Generating a Custom Warning, Error, or System Log Message" on page 43.)

The script is shown in both XSLT and SLAX syntax.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:if test="not(chassis/source-route)">
      <xnm:warning>
        <xsl:call-template name="jcs:edit-path">
          <xsl:with-param name="dot" select="chassis"/>
        </xsl:call-template>
        <message>IP source-route processing is not enabled.</message>
      </xnm:warning>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  if (not(chassis/source-route)) {
    <xnm:warning> {
      call jcs:edit-path($dot = chassis);
      <message> "IP source-route processing is not enabled.";
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

Download, enable, and test the script. To test that a commit script generates a warning message correctly, make sure that the candidate configuration contains the condition that elicits the warning. For this example, ensure that the **source-route** statement is not included at the **[edit chassis]** hierarchy level.

To test the example in this topic, perform the following steps:

1. Copy the XSLT or SLAX script into a text file, name the file **source-route.xsl** or **source-route.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts commit]** hierarchy level and **source-route.xsl** or **source-route.slax** as appropriate.

   ```
   [edit]
   user@host# set system scripts commit file source-route.xsl
   ```

3. If the **source-route** statement is included at the **[edit chassis]** hierarchy level, issue the **delete chassis source-route** configuration mode command:

   ```
   [edit]
   ```

```
user@host# delete chassis source-route
```

4. Issue the **commit and-quit** command.

```
[edit]
user@host# commit and-quit
```

## Verification

### Verifying Script Execution

**Purpose** Verify the warning message generated by the commit script.

**Action** Review the output of the **commit** command. The commit script generates a warning message when the **source-route** statement is not included at the **[edit chassis]** hierarchy level of the configuration. The warning does not affect the commit process.

```
[edit]
user@host# commit
[edit chassis]
  warning: IP source-route processing is not enabled.
commit complete
```

To display the XML-formatted version of the warning message, issue the **commit check | display xml** command:

```
[edit]

user@host# commit check | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
    <commit-results>
        <routing-engine junos:style="normal">
            <name>re0</name>
            <xnm:warning>
                <edit-path>
                    [edit chassis]
                </edit-path>
                <message>
                    IP source-route processing is not enabled.
                </message>
            </xnm:warning>
            <commit-check-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

To display a detailed trace of commit script processing, issue the **commit check | display detail** command:

```
[edit]

user@host# commit check | display detail
2009-06-15 14:40:29 PDT: reading commit script configuration
2009-06-15 14:40:29 PDT: testing commit script configuration
2009-06-15 14:40:29 PDT: opening commit script
'/var/db/scripts/commit/source-route-warning.xsl'
2009-06-15 14:40:29 PDT: reading commit script 'source-route-warning.xsl'
2009-06-15 14:40:29 PDT: running commit script 'source-route-warning.xsl'
2009-06-15 14:40:29 PDT: processing commit script 'source-route-warning.xsl'
```

```
[edit chassis]
    warning: IP source-route processing is not enabled.
2009-06-15 14:40:29 PDT: no errors from source-route-warning.xsl
2009-06-15 14:40:29 PDT: saving commit script changes
2009-06-15 14:40:29 PDT: summary: changes 0, transients 0 (allowed), syslog 0
2009-06-15 14:40:29 PDT: no commit script changes
2009-06-15 14:40:29 PDT: exporting juniper.conf
2009-06-15 14:40:29 PDT: expanding groups
2009-06-15 14:40:29 PDT: finished expanding groups
2009-06-15 14:40:29 PDT: setup foreign files
2009-06-15 14:40:29 PDT: propagating foreign files
2009-06-15 14:40:30 PDT: complete foreign files
2009-06-15 14:40:30 PDT: daemons checking new configuration
configuration check succeeds
```

**Related Documentation**

- Example: Generating a Custom Error Message on page 51
- Example: Generating a Custom System Log Message on page 54
- Generating a Custom Warning, Error, or System Log Message on page 43

## Example: Generating a Custom Error Message

This example commit script generates a custom error message when a specific statement is not included in the device configuration, thereby halting the commit operation.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

Using a commit script, write a custom error message that appears when the **description** statement is not included at the **[edit interfaces t1-*fpc/pic/port*]** hierarchy level:

The script is shown in both XSLT and SLAX syntax.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="interface" select="interfaces/interface"/>
    <xsl:for-each select="$interface[starts-with(name, 't1-')]">
      <xsl:variable name="ifname" select="."/>
      <xsl:if test="not(description)">
```

```
            <xnm:error>
              <xsl:call-template name="jcs:edit-path"/>
              <xsl:call-template name="jcs:statement"/>
              <message>Missing a description for this T1 interface.</message>
            </xnm:error>
          </xsl:if>
        </xsl:for-each>
      </xsl:template>
    </xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $interface = interfaces/interface;
  for-each ($interface[starts-with(name, 't1-')]) {
    var $ifname = .;
    if (not(description)) {
      <xnm:error> {
        call jcs:edit-path();
        call jcs:statement();
        <message> "Missing a description for this T1 interface.";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

Download, enable, and test the script: To test that a commit script generates an error message correctly, make sure that the candidate configuration contains the condition that elicits the error. For this example, ensure that the configuration for a T1 interface does not include the **description** statement.

To test the example in this topic, perform the following steps:

1. Copy the XSLT or SLAX script into a text file, name the file **description.xsl** or **description.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts commit]** hierarchy level and **description.xsl** or **description.slax** as appropriate.

   ```
   [edit]
   user@host# set system scripts commit file description.xsl
   ```

3. If the configuration for every T1 interface includes the **description** statement, issue the following configuration mode commands:

   ```
   [edit]
   user@host# edit interfaces t1-0/0/1
   [edit interfaces t1-0/0/1]
   user@host# delete description
   ```

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying Script Execution

**Purpose** Verify the error message generated by the commit script.

**Action** Review the output of the **commit** command. The commit script generates an error message for each T1 interface that does not include a **description** statement. Any error causes the commit process to fail.

```
[edit]
user@host# commit
[edit interfaces interface t1-0/0/1]
  'description'
    Missing a description for this T1 interface.
[edit interfaces interface t1-0/0/2]
  'description'
    Missing a description for this T1 interface.
error: 2 errors reported by commit scripts
error: commit script failure
```

To display the XML-formatted version of the error message, issue the **commit check | display xml** command:

```
[edit interfaces t1-0/0/1]

user@host# commit check | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
    <commit-results>
        <routing-engine junos:style="normal">
            <name>re0</name>
            <xnm:error>
                <edit-path>
                    [edit interfaces interface t1-0/0/1]
                </edit-path>
                <statement>
                    description
                </statement>
                <message>
                    Missing a description for this T1 interface.
                </message>
            </xnm:error>
            <xnm:error>
                <edit-path>
                    [edit interfaces interface t1-0/0/2]
                </edit-path>
                <statement>
                    description
                </statement>
                <message>
                    Missing a description for this T1 interface.
                </message>
            </xnm:error>
            <xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm"
```

```
                                    xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
                        <message>
                            2 errors reported by commit scripts
                        </message>
                    </xnm:error>
                    <xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm"
                  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
                        <message>
                            commit script failure
                        </message>
                    </xnm:error>
                </routing-engine>
            </commit-results>
            <cli>
                <banner>[edit interfaces]</banner>
            </cli>
        </rpc-reply>
```

To display a detailed trace of commit script processing, issue the **commit check | display detail** command:

```
[edit interfaces t1-0/0/1]

user@host# commit check | display detail
2009-06-15 15:56:09 PDT: reading commit script configuration
2009-06-15 15:56:09 PDT: testing commit script configuration
2009-06-15 15:56:09 PDT: opening commit script '/var/db/scripts/commit/error.xsl'
2009-06-15 15:56:09 PDT: reading commit script 'error.xsl'
2009-06-15 15:56:09 PDT: running commit script 'error.xsl'
2009-06-15 15:56:09 PDT: processing commit script 'error.xsl'
[edit interfaces interface t1-0/0/1]
    'description'
        Missing a description for this T1 interface.
[edit interfaces interface t1-0/0/2]
    'description'
        Missing a description for this T1 interface.
2009-06-15 15:56:09 PDT: 2 errors from script 'error.xsl'
error: 2 errors reported by commit scripts
error: commit script failure
```

**Related Documentation**

- Example: Generating a Custom System Log Message on page 54
- Example: Generating a Custom Warning Message on page 48
- Generating a Custom Warning, Error, or System Log Message on page 43

## Example: Generating a Custom System Log Message

This example commit script generates a custom system log message when a specific statement is not included in the device configuration.

- Requirements on page 55
- Overview and Commit Script on page 55
- Configuration on page 56
- Verification on page 56

## Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

Using a commit script, write a custom system log message that appears when the **read-write** statement is not included at the **[edit snmp community *community-name* authorization]** hierarchy level.

The script is shown in both XSLT and SLAX syntax.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="snmp/community">
      <xsl:if test="not(authorization/read-write)">
        <syslog>
          <message>SNMP community does not have read-write access.
          </message>
        </syslog>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (snmp/community) {
    if (not(authorization/read-write)) {
      <syslog> {
        <message> "SNMP community does not have read-write access.";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

Download, enable, and test the script. To test that a commit script generates a system log message correctly, make sure that the candidate configuration contains the condition that elicits the system log message. For this example, ensure that the **read-write** statement is not included at the **[edit snmp community *community-name* authorization]** hierarchy level.

To test the example in this topic, perform the following steps:

1. Copy the XSLT or SLAX script into a text file, name the file **read-write.xsl** or **read-write.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts commit]** hierarchy level and **read-write.xsl** or **read-write.slax** as appropriate.

   ```
   [edit]
   user@host# set system scripts commit file read-write.xsl
   ```

3. If the **read-write** statement is included at the **[edit snmp community *community-name* authorization]** hierarchy level, issue the following configuration mode command:

   ```
   [edit]
   user@host# delete snmp community community-name authorization read-write
   ```

4. Issue the following command to verify that system logging is configured to write to a file (a commonly used file name is **messages**):

   ```
   [edit]
   user@host# show system syslog
   ```

   For information about system log configuration, see the *Junos OS System Log Messages Reference*.

5. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying Script Execution

**Purpose**

Verify the system log message generated by the commit script.

**Action**

System log messages are generated during a commit operation but not during a commit check operation. This means you cannot use the **commit check | display xml** or **commit check | display detail** configuration mode commands to verify the output of system log messages. When the commit operation completes, inspect the system log file. The default directory for log files is **/var/log/**. View the log file by issuing the **file show *filename*** operational mode command. For example, if messages are logged to the **messages** file, issue the following command:

```
user@host> file show /var/log/messages
```

System log entries generated by commit scripts have the following format:

```
timestamp host-name cscript: message
```

Since the **read-write** statement was not included at the **[edit snmp community *community-name* authorization]** hierarchy level, the commit script should generate the "SNMP community does not have read-write access" message in the system log file.

```
Jun 3 14:34:37 host-name cscript: SNMP community does not have read-write access
```

**Related Documentation**

- Example: Generating a Custom Error Message on page 51
- Example: Generating a Custom Warning Message on page 48
- Generating a Custom Warning, Error, or System Log Message on page 43

CHAPTER 7

# Generating a Persistent or Transient Configuration Changes



## Generating a Persistent or Transient Change

To generate a persistent or transient change, follow these steps:

1. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from "Required Boilerplate for Commit Scripts" on page 30. It is reproduced here for convenience:

### XSLT Boilerplate

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <!-- ... Insert your code here ... -->
  </xsl:template>
</xsl:stylesheet>
```

### SLAX Boilerplate

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
```

```
match configuration {
  /*
  * Insert your code here
  */
}
```

2. At the position indicated by the comment "***Insert your code here***," include one or more XSLT programming instructions or their SLAX equivalents. Commonly used XSLT constructs include the following.

   - **<xsl:choose> <xsl:when> <xsl:otherwise>**—Conditional construct that causes different instructions to be processed in different circumstances. The **<xsl:choose>** instruction contains one or more **<xsl:when>** elements, each of which tests an XPath expression. If the test evaluates as true, the XSLT processor executes the instructions in the **<xsl:when>** element. The XSLT processor processes only the instructions contained in the first **<xsl:when>** element whose **test** attribute evaluates as true. If none of the **<xsl:when>** elements' **test** attributes evaluate as true, the content of the **<xsl:otherwise>** element, if there is one, is processed.

   - **<xsl:for-each select="*xpath-expression*">**—Programming instruction that tells the XSLT processor to gather together a set of nodes and process them one by one. The nodes are selected by the Extensible Markup Language (XML) Path Language (XPath) expression in the **select** attribute. Each of the nodes is then processed according to the instructions contained in the **<xsl:for-each>** instruction. Code inside an **<xsl:for-each>** instruction is evaluated recursively for each node that matches the XPath expression. The context is moved to the node during each pass.

   - **<xsl:if test="*xpath-expression*">**—Conditional construct that causes instructions to be processed if the XPath expression in the **test** attribute evaluates to **true**.

   For example, the following XSLT programming instructions select each SONET/SDH interface that does not have the MPLS protocol family enabled:

   ```
   <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]/unit">
     <xsl:if test="not(family/mpls)">
   ```

   In SLAX, the **for-each** and **if** constructs look like this:

   ```
   for-each (interfaces/interface[starts-with(name, 'so-')]/unit) {
     if (not(family/mpls)) {
   ```

   For more information about how to use programming instructions, including examples and pseudocode, see XSLT Programming Instructions Overview. For information about writing scripts in SLAX instead of XSLT, see SLAX Overview.

3. Include instructions for changing the configuration. There are two ways to generate a persistent change and two ways to generate a transient change. To generate a persistent change, you can either reference the **jcs:emit-change** template or include a **<change>** element. To generate a transient change, you can either reference the **jcs:emit-change** template and pass in the **tag** parameter with **'transient-change'** selected or include a **<transient-change>** element.

   The **jcs:emit-change** template allows for more efficient, less error-prone scripting because you can define the content of the change without specifying the complete

XML hierarchy for the affected statement. Instead, the XML hierarchy is defined in the XPath expression contained in the script's programming instruction.

Consider the following examples. Both of the persistent change examples have the same result, even though they place the **unit** statement in different locations in the **<xsl:for-each>** and **<xsl:if>** programming instructions. In both cases, the script searches for SONET/SDH interfaces that do not have the MPLS protocol family enabled, adds the **family mpls** statement at the **[edit interfaces so-*fpc/pic/port* unit *logical-unit-number*]** hierarchy level, and emits a warning message stating that the configuration has been changed. Likewise, both of the transient change examples have the same result. They both set Point-to-Point Protocol (PPP) encapsulation on all SONET/SDH interface that have IP version 4 (IPv4) enabled.

### Persistent Change Generated with the jcs:emit-change Template

In this example, the content of the persistent change (contained in the **content** parameter) is specified without including the complete XML hierarchy. Instead, the XPath expression in the **<xsl:for-each>** programming instruction sets the context for the change.

The message parameter is also included. This parameter causes the **jcs:emit-change** template to call the **<xnm:warning>** template, which sends a warning notification to the CLI. The message parameter automatically includes the current hierarchy information in the warning message. (For more information about the parameters available with the **jcs:emit-change** template, see jcs:emit-change Template.)

```
<xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]/unit">
  <xsl:if test="not(family/mpls)">
    <xsl:call-template name="jcs:emit-change">
      <xsl:with-param name="content">
          <family>
            <mpls/>
          </family>
      </xsl:with-param>
      <xsl:with-param name="message">
        <xsl:text>Adding 'family mpls' to SONET interface.</xsl:text>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:if>
</xsl:for-each>
```

### Persistent Change Generated with the <change> Element

In this example, the complete XML hierarchy leading to the affected statement must be included as child elements of the **<change>** element.

This example includes the current hierarchy information in the warning message by referencing the **jcs:edit-path** and **jcs:statement** templates. For more information about warning messages, see "Overview of Generating Custom Warning, Error, and System Log Messages" on page 11.

```
<xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]">
  <xsl:if test="not(unit/family/mpls)">
    <change>
      <interfaces>
        <interface>
```

```
            <name><xsl:value-of select="name"/></name>
            <unit>
              <name><xsl:value-of select="unit/name"/></name>
              <family>
                <mpls/>
              </family>
            </unit>
          </interface>
        </interfaces>
      </change>
      <xnm:warning>
        <xsl:call-template name="jcs:edit-path"/>
        <xsl:call-template name="jcs:statement">
          <xsl:with-param name="dot" select="unit/name"/>
        </xsl:call-template>
        <message>Adding 'family mpls' to SONET interface.</message>
      </xnm:warning>
    </xsl:if>
  </xsl:for-each>
```

### Transient Change Generated with the jcs:emit-change Template

In this example, the content of the transient change (contained in the **content** parameter) is specified without including the complete XML hierarchy. Instead, the XPath expression in the **<xsl:for-each>** programming instruction sets the context of the change. The **and** operator in the XPath expression means both operands are **true** when converted to Booleans; the second operand is not evaluated if the first operand is **false**.

The tag parameter is included with '**transient-change**' selected. Without the **tag** parameter, the **jcs:emit-change** template generates a persistent change by default. (For more information about the parameters available with the **jcs:emit-change** template, see jcs:emit-change Template.)

```
<xsl:for-each select="interfaces/interface[starts-with(name, 'so-') \
            and unit/family/inet]">
  <xsl:call-template name="jcs:emit-change">
    <xsl:with-param name="tag" select="'transient-change'"/>
    <xsl:with-param name="content">
      <encapsulation>ppp</encapsulation>
    </xsl:with-param>
  </xsl:call-template>
</xsl:for-each>
```

### Transient Change Generated with the <transient-change> Element

In this example, the complete XML hierarchy leading to the affected statement must be included as child elements of the **<transient-change>** element.

```
<xsl:for-each select="interfaces/interface[starts-with(name, 'so-')\
            and unit/family/inet]">
  <transient-change>
    <interfaces>
      <interface>
        <name><xsl:value-of select="name"/></name>
        <encapsulation>ppp</encapsulation>
      </interface>
```

```
        </interfaces>
      </transient-change>
    </xsl:for-each>
```

4. Save the script with a meaningful name.
5. Copy the script to either the **/var/db/scripts/commit** directory on the device hard drive or the **/config/scripts/commit** directory on the flash drive. For information about setting the storage location for commit scripts, see Storing Scripts in Flash Memory.

   If the device has dual Routing Engines and you want the script to take effect on both of them, you must copy the script to the **/var/db/scripts/commit** or the **/config/scripts/commit** directory on both Routing Engines. The **commit synchronize** command does not copy scripts between Routing Engines.
6. Enable the script by including the **file** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   file filename;
   ```

   where ***filename*** is the name you assigned in Step 4.
7. If the script makes transient changes, include the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   allow-transients;
   ```

If all the commit scripts run without errors, any transient changes are loaded into the checkout configuration, but not to the candidate configuration. Any persistent changes are loaded into the candidate configuration. The commit process then continues by validating the configuration and propagating changes to the affected processes on the device.

To display the configuration with both persistent and transient changes applied, issue the **show | display commit-scripts** configuration mode command:

```
[edit]
user@host# show | display commit-scripts
```

To display the configuration with only persistent changes applied, issue the **show | display commit-scripts no-transients** configuration mode command:

```
[edit]
user@host# show | display commit-scripts no-transients
```

Persistent and transient changes are loaded into the configuration in the same manner that the **load replace** configuration mode command loads an incoming configuration. When generating a persistent or transient change, adding the **replace="replace"** attribute to a configuration element produces the same behavior as a **replace:** tag in a **load replace** operation. Both persistent and transient changes are loaded into the configuration with the **load replace** behavior, but persistent changes are loaded into the candidate configuration and transient changes are loaded into the checkout configuration.

## Removing a Persistent or Transient Change

After a commit script changes the configuration, you can remove the change and return the configuration to its previous state.

For persistent changes only, you can undo the configuration change by issuing the **delete**, **deactivate**, or **rollback** configuration mode command and committing the configuration. For both persistent and transient changes, you must remove, delete, or deactivate the associated commit script, or else the commit script regenerates the change during a subsequent commit operation.

Deleting the **file *filename*** statement from the configuration effectively "unconfigures" the functionality associated with the corresponding commit script. Deactivating the statement adds the **inactive:** tag to the statement, effectively commenting out the statement from the configuration. Statements marked as inactive do not take effect when you issue the **commit** command.

To reverse the effect of a commit script and prevent the script from running again, perform the following steps:

1. For persistent changes only, delete or deactivate the statement that was added by the commit script:

   ```
   [edit]
   user@host# delete (statement | identifier)
   - OR –
   user@host# deactivate (statement | identifier)
   ```

   Alternatively, you can roll back the configuration to a candidate that does not contain the statement.

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

2. Either delete or deactivate the commit script, or remove or comment out the section of code that generates the unwanted change. To delete or deactivate the script, issue one of the following commands.

   ```
   [edit]
   user@host# delete system scripts commit file filename
   - OR –
   user@host# deactivate system scripts commit file filename
   ```

3. Issue the **commit** command:

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

4. If you are deleting the reference to the script from the configuration, you can also remove the file from commit scripts storage directory (either **/var/db/scripts/commit** on the hard drive or **/config/scripts/commit** on the flash drive; for information about setting the storage location for commit scripts, see Storing Scripts in Flash Memory.) To do this, exit configuration mode and issue the **file delete** operational mode command:

   ```
   [edit]
   ```

```
user@host# exit

user@host> file delete /var/db/scripts/commit/filename
- OR –
user@host> file delete /config/scripts/commit/filename
```

## Tag Elements to Use When Generating Persistent and Transient Changes

Table 4 on page 65 describes the data that you can include in the **<change>** tag element in a commit script. To see how data values are supplied within a script, see Examples: Generating Persistent and Transient Changes.

### Table 4: Tags and Attributes for Creating Configuration Changes

| Data Item, XML Element, or Attribute | Description |
|---|---|
| **Container Tags** | |
| **<change>** | Request that the Junos XML protocol server load configuration data into the candidate configuration. |
| **<transient-change>** | Request that the Junos XML protocol server load configuration data into the configuration. |
| **Content Tags** | |
| **<jcs:emit-change>** | This is a template in the file **junos.xsl**. This template converts the contents of the **<xsl:with-param>** element into a **<change>** request. |
| **<xsl:with-param name="content">** | You use the **content** parameter with the **jcs:emit-change** template. It allows you to include the content of the change, relative to **dot**. |
| **<xsl:with-param name="tag" select="'transient-change'"/>** | Convert the contents of the **content** parameter into a **<transient-change>** request.<br><br>You use the **tag** parameter with the **jcs:emit-change** template.<br><br>By default, the **jcs:emit-change** template converts the contents of the **content** parameter into a **<change>** (persistent change) request. |

## Example: Generating a Persistent Change

If you do not explicitly configure the MPLS protocol family on an interface, the interface is not enabled for MPLS applications. This example generates a persistent change that adds the **family mpls** statement in the configuration of SONET/SDH interfaces when the statement is not already included in the configuration.

- Requirements on page 66
- Overview and Commit Script on page 66

## Requirements

This example uses a device running Junos OS with one or more SONET/SDH interfaces.

## Overview and Commit Script

The commit script in this example finds all SONET/SDH interfaces that have a logical interface configured but that do not have the **family mpls** statement configured. For these interfaces, the script adds the **family mpls** statement to the interface configuration as a persistent change at the **[edit interfaces** *interface-name* **unit** *logical-unit-number* hierarchy level.

The persistent change is generated by the **jcs:emit-change** template, which is a helper template contained in the **junos.xsl** import file. The **tag** parameter of the **jcs:emit-change** template is omitted, which directs the script to emit the change as a persistent change. The **content** parameter of the **jcs:emit-change** template includes the configuration statements to be added as a persistent change. The **message** parameter of the **jcs:emit-change** template includes the warning message to be displayed in the CLI, notifying you that the configuration has been changed.

The script is shown in both XSLT and SLAX syntax.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]/unit">
      <xsl:if test="not(family/mpls)">
        <xsl:call-template name="jcs:emit-change">
          <xsl:with-param name="message">
            <xsl:text>Adding 'family mpls' to SONET/SDH interface.</xsl:text>
          </xsl:with-param>
          <xsl:with-param name="content">
              <family>
                <mpls/>
              </family>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
```

```
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/interface[starts-with(name, 'so-')]/unit) {
    if (not(family/mpls)) {
      call jcs:emit-change() {
        with $message = {
          expr "Adding 'family mpls' to SONET/SDH interface.";
        }
        with $content = {
          <family> {
            <mpls>;
          }
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

Download, enable, and test the script.

1. Copy the XSLT or SLAX script into a text file, name the file **mpls.xsl** or **mpls.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. In configuration mode, include the **file** statement at the **[edit system scripts commit]** hierarchy level and **mpls.xsl** or **mpls.slax** as appropriate.

   ```
   [edit]
   user@host# set system scripts commit file mpls.xsl
   ```

3. To test that the commit script generates the persistent change correctly, make sure that the configuration contains the condition that elicits the change. To test this script, ensure that the **family mpls** statement is not included at the **[edit interfaces so-*fpc/pic/port* unit *logical-unit-number*]** hierarchy level for at least one SONET/SDH interface.

   If the **family mpls** statement is included at the **[edit interfaces so-*fpc/pic/port* unit *logical-unit-number*]** hierarchy level, issue the following configuration mode command to delete the statement:

   ```
   [edit]
   user@host# delete interfaces so-fpc/pic/port unit logical-unit-number family mpls
   ```

4. The **commit check** command verifies the syntax of the configuration prior to a commit, but it does not commit the changes. The commit script in this example produces a message for each change it makes. Use the **commit check** command to preview these messages to determine whether the script will update the configuration with the **family mpls** statement for the appropriate interfaces.

   Issue the **commit check | display xml** command to display the XML-formatted version of the message. The sample output indicates that the script will add the **family mpls** statement to the so-2/3/4 interface configuration during the commit operation.

   ```
   [edit]
   ```

```
user@host# commit check | display xml

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.2R1/junos">
    <commit-results>
        <routing-engine junos:style="normal">
            <name>re0</name>
            <xnm:warning xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
                <edit-path>
                    [edit interfaces interface so-2/3/4 unit 0]
                </edit-path>
                <message>
                    Adding 'family mpls' to SONET/SDH interface.
                </message>
            </xnm:warning>
            <commit-check-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

5. To display a detailed trace of commit script processing, issue the **commit check | display detail** command. In the sample output, there is one persistent change that will be loaded into the configuration during the commit operation.

```
[edit]

user@host# commit check | display detail

2011-06-17 14:17:35 PDT: reading commit script configuration
2011-06-17 14:17:35 PDT: testing commit script configuration
2011-06-17 14:17:35 PDT: opening commit script
'/var/db/scripts/commit/mpls.xsl'
2011-06-17 14:17:35 PDT: reading commit script 'mpls.xsl'
2011-06-17 14:17:35 PDT: running commit script 'mpls.xsl'
2011-06-17 14:17:35 PDT: processing commit script 'mpls.xsl'
2011-06-17 14:17:35 PDT: no errors from mpls.xsl
2011-06-17 14:17:35 PDT: saving commit script changes for script mpls.xsl
2011-06-17 14:17:35 PDT: summary of script mpls.xsl: changes 1, transients
 0, syslog 0
2011-06-17 14:17:35 PDT: start loading commit script changes
2011-06-17 14:17:35 PDT: loading commit script changes into real db
2011-06-17 14:17:35 PDT: finished commit script changes into real db
2011-06-17 14:17:35 PDT: no transient commit script changes
2011-06-17 14:17:35 PDT: finished loading commit script changes
2011-06-17 14:17:35 PDT: copying juniper.db to juniper.data+
2011-06-17 14:17:35 PDT: finished copying juniper.db to juniper.data+
...
configuration check succeeds
```

6. After verifying that the script produces the correct changes, issue the **commit** command to start the commit operation and execute the script.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the correct changes are integrated into the configuration.

**Action** After executing the commit operation, view the configuration by issuing the **show interfaces** configuration mode command. If the MPLS protocol family is not enabled on one or more SONET/SDH interfaces before the script runs, the output is similar to the following:

```
[edit]

user@host# show interfaces
... other configured interface types ...
so-2/3/4 {
    unit 0 {
        family mpls; # Added by persistent change
    }
}
... other configured interface types ...
```

**Related Documentation**

- Example: Generating a Transient Change on page 69
- Generating a Persistent or Transient Change on page 59
- Overview of Generating Persistent or Transient Configuration Changes on page 13
- Removing a Persistent or Transient Change on page 64
- jcs:emit-change Template

## Example: Generating a Transient Change

This example uses a commit script to set PPP encapsulation on all SONET/SDH interfaces with the IPv4 protocol family enabled. The changes are added as transient changes.



### Requirements

This example uses a device running Junos OS with one or more SONET/SDH interfaces.

### Overview and Commit Script

The commit script in this example finds all SONET/SDH interfaces with the IPv4 protocol family enabled in the configuration and adds the **encapsulation ppp** statement to the interface configuration. The transient change is generated by the **jcs:emit-change** template, which is a helper template contained in the **junos.xsl** import file. The **tag** parameter of the **jcs:emit-change** template has the value **transient-change**, which directs the script to emit the change as a transient change rather than a persistent change. The **content** parameter of the **jcs:emit-change** template includes the configuration statements to be added as a transient change.

The script is shown in both XSLT and SLAX syntax.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')
            and unit/family/inet]">
      <xsl:call-template name="jcs:emit-change">
        <xsl:with-param name="tag" select="'transient-change'"/>
        <xsl:with-param name="content">
          <encapsulation>ppp</encapsulation>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/interface[starts-with(name, 'so-') and unit/family/inet]) {
    call jcs:emit-change($tag = 'transient-change') {
      with $content = {
        <encapsulation> "ppp";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

Download, enable, and test the script.

1. Copy the XSLT or SLAX script into a text file, name the file **encap-ppp.xsl** or **encap-ppp.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. In configuration mode, include the **file** statement at the **[edit system scripts commit]** hierarchy level and **encap-ppp.xsl** or **encap-ppp.slax** as appropriate.

   ```
   [edit]
   user@host# set system scripts commit file encap-ppp.xsl
   ```

3. Include the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level. This enables commit scripts to load transient changes into the checkout configuration.

   ```
   [edit]
   ```

```
user@host# set system scripts commit allow-transients
```

4. To test that the commit script generates the transient change correctly, make sure that the configuration contains the condition that elicits the change. Ensure that the **encapsulation ppp** statement is not included at the **[edit interfaces so-*fpc/pic/port*]** hierarchy level for at least one SONET/SDH interface.

   If the **encapsulation ppp** statement is included at the **[edit interfaces so-*fpc/pic/port*]** hierarchy level, issue the following configuration mode command to delete the statement:

   ```
   [edit]
   user@host# delete interfaces so-fpc/pic/port encapsulation ppp
   ```

5. The **commit check** command verifies the syntax of the configuration prior to a commit, but it does not commit the changes. Issue the **commit check** command to preview a trace of commit script processing to verify that the script will add the transient change to the checkout configuration.

   Issue the **commit check | display detail** command to display a detailed trace of commit script processing. In the sample output, there are two transient changes that are loaded into the checkout configuration.

   ```
   [edit]

   user@host# commit check | display detail

   2011-06-15 12:07:30 PDT: reading commit script configuration
   2011-06-15 12:07:30 PDT: testing commit script configuration
   2011-06-15 12:07:30 PDT: opening commit script
   '/var/db/scripts/commit/encap-ppp.xsl'
   2011-06-15 12:07:30 PDT: reading commit script 'encap-ppp.xsl'
   2011-06-15 12:07:30 PDT: running commit script 'encap-ppp.xsl'
   2011-06-15 12:07:30 PDT: processing commit script 'encap-ppp.xsl'
   2011-06-15 12:07:30 PDT: no errors from encap-ppp.xsl
   2011-06-15 12:07:30 PDT: saving commit script changes for script
   encap-ppp.xsl
   2011-06-15 12:07:30 PDT: summary of script encap-ppp.xsl: changes 0,
   transients 2 (allowed), syslog 0
   2011-06-15 12:07:30 PDT: start loading commit script changes
   2011-06-15 12:07:30 PDT: no commit script changes
   2011-06-15 12:07:30 PDT: updating transient changes into transient tree
   2011-06-15 12:07:30 PDT: finished loading commit script changes
   2011-06-15 12:07:30 PDT: copying juniper.db to juniper.data+
   2011-06-15 12:07:30 PDT: finished copying juniper.db to juniper.data+
   2011-06-15 12:07:30 PDT: exporting juniper.conf
   2011-06-15 12:07:30 PDT: merging transient changes
   ...
   configuration check succeeds
   ```

6. After verifying that the script produces the correct changes, issue the **commit** command to start the commit operation and execute the script.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the correct changes are integrated into the checkout configuration. If there are one or more SONET/SDH interfaces with the IPv4 protocol family enabled, you should see the **encapsulation ppp** statement added as a transient change to the interface hierarchy.

**Action** To view the configuration with transient changes, issue the **show interfaces | display commit-scripts** configuration mode command. The **show interfaces | display commit-scripts** command displays all the statements that are in the configuration, including statements that are generated by transient changes. If there are one or more SONET/SDH interfaces with the IPv4 protocol family enabled, the output is similar to the following:

```
[edit]

user@host# show interfaces | display commit-scripts
... other configured interface types ...
so-1/2/3 {
    mtu 576;
    encapsulation ppp; /* Added by transient change. */
    unit 0 {
        family inet {
            address 10.0.0.3/32;
        }
    }
}
so-1/2/4 {
    encapsulation ppp; /* Added by transient change. */
    unit 0 {
        family inet {
            address 10.0.0.4/32;
        }
    }
}
so-2/3/4 {
   encapsulation cisco-hdlc; # Not affected by this script, because IPv4 protocol
                             # family is not configured on this interface.
    unit 0 {
        family mpls;
    }
}
... other configured interface types ...
```

## Troubleshooting

### Troubleshooting Commit Errors

**Problem** The CLI generates an invalid transient change error, and the commit fails.

```
user@host# commit check
error: invalid transient change generated by commit script: encap-ppp.xsl
warning: 1 transient change was generated without [system scripts commit
allow-transients]
```

```
error: 1 error reported by commit scripts
error: commit script failure
```

**Solution** You must configure the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level to enable commit scripts to load transient changes into the checkout configuration.

Issue the following configuration mode command to allow transient changes:

```
[edit]
user@host# set system scripts commit allow-transients
```

**Related Documentation**

- Example: Generating a Persistent Change on page 65
- Generating a Persistent or Transient Change on page 59
- Overview of Generating Persistent or Transient Configuration Changes on page 13
- Removing a Persistent or Transient Change on page 64
- jcs:emit-change Template

CHAPTER 8

# Creating Custom Configuration Syntax with Macros



## Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements

By itself, the custom syntax in an **apply-macro** statement has no operational impact on the device. To give meaning to your syntax, there must be a corresponding commit script that uses the syntax as data for generating related standard Junos OS statements. To write such a script, follow these steps:

1. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from "Required Boilerplate for Commit Scripts" on page 30. It is reproduced here for convenience:

### XSLT Boilerplate

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <!-- ... Insert your code here ... -->
  </xsl:template>
</xsl:stylesheet>
```

### SLAX Boilerplate

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
```

```
match configuration {
  /*
  * Insert your code here
  */
}
```

2. At the position indicated by the comment "***Insert your code here***," include XSLT programming instructions (or their SLAX equivalents) that inspect the configuration for the **apply-macro** statement at a specified hierarchy level and change the configuration to include standard Junos OS CLI syntax.

   For an example that uses both types of instructions and includes a line-by-line analysis of the XSLT syntax, see "Example: Creating Custom Configuration Syntax with Macros" on page 77.

3. Save the script with a meaningful name.

4. Copy the script to either the **/var/db/scripts/commit** directory on the hard drive or the **/config/scripts/commit** directory on the flash drive. For information about setting the storage location for commit scripts, see Storing Scripts in Flash Memory.

   If the device has dual Routing Engines and you want the script to take effect on both of them, you must copy the script to the **/var/db/scripts/commit** or the **/config/scripts/commit** directory on both Routing Engines. The **commit synchronize** command does not copy scripts between Routing Engines.

5. Enable the script by including the **file** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   file filename;
   ```

   where ***filename*** is the name of the script.

6. If the script makes transient changes, include the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   allow-transients;
   ```

If all the commit scripts run without errors, any transient changes are loaded into the checkout configuration, but not to the candidate configuration. Any persistent changes are loaded into the candidate configuration. The commit process then continues by validating the configuration and propagating changes to the affected processes on the device running Junos OS.

**Related Documentation**

- Overview of Creating Custom Configuration Syntax with Macros on page 19
- How Macros Work on page 19
- Example: Creating Custom Configuration Syntax with Macros on page 77

## Example: Creating Custom Configuration Syntax with Macros

This commit script example shows how to create custom configuration syntax using macros.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

Figure 6 on page 77 shows a macro that uses custom syntax and the corresponding expansion to standard Junos OS command-line interface (CLI) syntax.

#### Figure 6: Sample Macro and Corresponding Junos OS CLI Expansion

**Custom macro syntax:**

```
protocols {
    mpls {
        apply-macro  blue-type-lsp {
            10.1.1.1;
            10.1.1.2;
            10.1.1.3;
            10.1.1.4;
            color blue;
        }
    }
}
```

**macro expansion** →

**Expanded JUNOS CLI syntax:**

```
protocols {
    mpls {
        apply-macro  blue-type-lsp { ... }
        label-switched-path blue-lsp-10.1.1.1 {
            to 10.1.1.1;
            admin-group include-any blue;
        }
        label-switched-path blue-lsp-10.2.2.2 {
            to 10.2.2.2;
            admin-group include-any blue;
        }
        label-switched-path blue-lsp-10.3.3.3 {
            to 10.3.3.3;
            admin-group include-any blue;
        }
        label-switched-path blue-lsp-10.4.4.4 {
            to 10.4.4.4;
            admin-group include-any blue;
        }
    }
}
```

g017029

In this example, the Junos OS management (mgd) process inspects the configuration, looking for **apply-macro** statements. For each **apply-macro** statement with the **color** parameter included at the **[edit protocols mpls]** hierarchy level, the script generates a transient change, using the data provided within the **apply-macro** statement to expand the macro into a standard Junos OS administrative group for LSPs.

For this example to work, an **apply-macro** statement must be included at the **[edit protocols mpls]** hierarchy level with a set of addresses, a **color**, and a **group-value** parameter. The commit script converts each address to an LSP configuration, and the script converts the **color** parameter into an administrative group.

Following are the commit script instructions that expand the macro in Figure 6 on page 77 and a line-by-line explanation of the script:

**XSLT Syntax**

```
1  <?xml version="1.0" standalone="yes"?>
2  <xsl:stylesheet version="1.0"
3     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4     xmlns:junos="http://xml.juniper.net/junos/*/junos"
5     xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6     xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
7     <xsl:import href="../import/junos.xsl"/>

8     <xsl:template match="configuration">
9       <xsl:variable name="mpls" select="protocols/mpls"/>
10      <xsl:for-each select="$mpls/apply-macro[data/name = 'color']">
11       <xsl:variable name="color" select="data[name = 'color']/value"/>
12       <xsl:for-each select="$mpls/apply-macro[data/name = 'group-value']">
13         <xsl:variable name="group-value" select="data[name = \
               'group-value']/value"/>
14         <transient-change>
15           <protocols>
16             <mpls>
17               <admin-groups>
18                 <name>
19                   <xsl:value-of select="$color"/>
20                 </name>
21                 <group-value>
22                   <xsl:value-of select="$group-value"/>
23                 </group-value>
24               </admin-groups>
25               <xsl:for-each select="data[not(value)]/name">
26                 <label-switched-path>
27                   <name>
28                     <xsl:value-of select="concat($color, '-lsp-', .)"/>
29                   </name>
30                   <to><xsl:value-of select="."/></to>
31                   <admin-group>
32                     <include-any>
33                       <xsl:value-of select="$color"/>
34                     </include-any>
35                   </admin-group>
36                 </label-switched-path>
37               </xsl:for-each>
38             </mpls>
39           </protocols>
40         </transient-change>
41       </xsl:for-each>
42     </xsl:for-each>
43   </xsl:template>
44 </xsl:stylesheet>
```

Lines 1 through 8 (and Lines 43 and 44) are the boilerplate that you include in every commit script. For brevity, Lines 1 through 8 are omitted here.

Line 9 assigns the **[edit protocols mpls]** hierarchy level to a variable called **$mpls**.

```
9   <xsl:variable name="mpls" select="protocols/mpls"/>
```

Line 10 selects every **apply-macro** statement at the **[edit protocols mpls]** hierarchy level that contains the **color** parameter. The sample configuration in Figure 6 on page 77 contains only one **apply-macro** statement. Therefore, this **<xsl:for-each>** programming instruction takes effect only once.

```
10    <xsl:for-each select="$mpls/apply-macro[data/name = 'color']">
```

Line 11 assigns the value of the **color** parameter, in this case **blue**, to a variable called **$color**.

```
11    <xsl:variable name="color" select="data[name = 'color']/value"/>
```

Line 12 selects every **apply-macro** statement at the **[edit protocols mpls]** hierarchy level that contains the **color** parameter. The sample configuration in Figure 6 on page 77 contains only one **apply-macro** statement. Therefore, this **<xsl:for-each>** programming instruction takes effect only once.

```
12    <xsl:for-each select="$mpls/apply-macro[data/name = 'color']">
```

Line 13 assigns the value of the **group-value** parameter, in this case **0**, to a variable called **$group-value**.

```
13     <xsl:variable name="group-value" select="data[name = 'group-value']/value"/>
```

Lines 14 through 16 generate a transient change at the **[edit protocols mpls]** hierarchy level.

```
14      <transient-change>
15        <protocols>
16          <mpls>
```

Lines 17 through 24 add the **admin-groups** statement to the configuration and assign the value of the **$color** variable to the group name and the value of the **$group-value** variable to the group value.

```
17              <admin-groups>
18                <name>
19                  <xsl:value-of select="$color"/>
20                </name>
21                <group-value>
22                  <<xsl:value-of select="$group-value"/>
23                </group-value>
24              </admin-groups>
```

The resulting configuration statements are as follows:

```
admin-groups {
  blue 0;
}
```

Line 25 selects the name of every parameter that does not already have a value assigned to it, which in this case are the four IP addresses. This **<xsl:for-each>** programming instruction uses recursion through the macro and selects each IP address in turn. The **color** and **group-value** parameters each already have a value assigned (**blue** and **0**, respectively), so this line does not apply to them.

```
25              <xsl:for-each select="data[not(value)]/name">
```

Line 26 adds the **label-switched-path** statement in the configuration.

```
26                <label-switched-path>
```

Lines 27 through 29 assign the **label-switched-path** a name that concatenates the value of the **$color** variable, the text **-lsp-**, and the current IP address currently selected by Line 25 (represented by the " **.** " ).

```
27              <name>
28                <xsl:value-of select="concat($color, '-lsp-', .)"/>
29              </name>
```

Line 30 adds the **to** statement to the configuration and sets its value to the IP address currently selected by Line 25.

```
30              <to><xsl:value-of select="."/></to>
```

Lines 31 through 35 add the **admin-group include-any** statement to the configuration and sets its value to the value of the **$color** variable.

```
31              <admin-group>
32                  <include-any>
33                    <xsl:value-of select="$color"/>
34                  </include-any>
35                </admin-group>
```

The resulting configuration statements (for one pass) are as follows:

```
label-switched-path blue-lsp-10.1.1.1 {
  to 10.1.1.1;
  admin-group include-any blue;
}
```

Lines 36 through 42 are closing tags.

```
36                </label-switched-path>
37              </xsl:for-each>
38            </mpls>
39          </protocols>
40        </transient-change>
41      </xsl:for-each>
42    </xsl:for-each>
```

Lines 43 and 44 are closing tags for Lines 8 and 2, respectively.

```
43   </xsl:template>
44 </xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $mpls = protocols/mpls;
  for-each ($mpls/apply-macro[data/name = 'color']) {
    var $color = data[name = 'color']/value;
    for-each ($mpls/apply-macro[data/name = 'group-value']) {
      var $group-value = data[name='group-value']/value;
      <transient-change> {
        <protocols> {
          <mpls> {
```

```
                <admin-groups> {
                  <name> $color;
                  <group-value> $group-value;
                }
                for-each (data[not(value)]/name) {
                  <label-switched-path> {
                    <name> $color _ '-lsp-' _ .;
                    <to> .;
                    <admin-group> {
                      <include-any> $color;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

For more information about this example, see "Example: Configuring Administrative Groups for LSPs" on page 104.

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **lsp-admin.xsl** or **lsp-admin.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **lsp-admin.slax**.

```
system {
  scripts {
    commit {
      allow-transients;
      file lsp-admin.xsl;
    }
  }
}
protocols {
  mpls {
    apply-macro blue-type-lsp {
      10.1.1.1;
      10.2.2.2;
      10.3.3.3;
      10.4.4.4;
      color blue;
      group-value 0;
    }
  }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** To display the configuration statements created by the script, issue the **show protocols mpls | display commit-scripts** command:

```
[edit]
user@host# show protocols mpls | display commit-scripts
apply-macro blue-type-lsp {
   10.1.1.1;
   10.2.2.2;
   10.3.3.3;
   10.4.4.4;
   color blue;
   group-value 0;
}
admin-groups {
   blue 0;
}
label-switched-path blue-lsp-10.1.1.1 {
   to 10.1.1.1;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.2.2.2 {
   to 10.2.2.2;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.3.3.3 {
   to 10.3.3.3;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.4.4.4 {
   to 10.4.4.4;
   admin-group include-any blue;
```

```
}
```

**Related Documentation**

- Overview of Creating Custom Configuration Syntax with Macros on page 19
- How Macros Work on page 19
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 75

CHAPTER 9

# Commit Script Examples



## Example: Adding a Final then accept Term to a Firewall

This commit script example adds a **then accept** statement to any firewall filter that does not already end with an explicit **then accept** statement.

## Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

Each firewall filter in Junos OS has an implicit discard action at the end of the filter, which is equivalent to the following explicit filter term:

```
term implicit-rule {
  then discard;
}
```

As a result, if a packet matches none of the terms in the filter, it is discarded. In some cases, you might want to override the default by adding a last term to accept all packets that do not match a firewall filter's series of match conditions. In this example, the commit script adds a final **then accept** statement to any firewall filter that does not already end with an explicit **then accept** statement.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:apply-templates select="firewall/filter | firewall/family/inet
          | firewall/family/inet6" mode="filter"/>
  </xsl:template>
  <xsl:template match="filter" mode="filter">
    <xsl:param name="last" select="term[position() = last()]"/>
    <xsl:comment>
      <xsl:text>Found </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text>; last </xsl:text>
      <xsl:value-of select="$last/name"/>
    </xsl:comment>
    <xsl:if test="$last and ($last/from or $last/to or not($last/then/accept))">
      <xnm:warning>
        <xsl:call-template name="jcs:edit-path"/>
        <message>
          <xsl:text>filter is missing final 'then accept' rule</xsl:text>
        </message>
      </xnm:warning>
      <xsl:call-template name="jcs:emit-change">
        <xsl:with-param name="content">
          <term>
            <name>very-last</name>
            <junos:comment>
              <xsl:text>This term was added by a commit script</xsl:text>
            </junos:comment>
```

```
            <then>
              <accept/>
            </then>
          </term>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  apply-templates firewall/filter | firewall/family/inet | firewall/family/inet6 {
    mode "filter";
  }
}
match filter {
  mode "filter";
  param $last = term[position() = last()];
  <xsl:comment> {
    expr "Found ";
    expr name;
    expr "; last ";
    expr $last/name;
  }
  if ($last and ($last/from or $last/to or not($last/then/accept))) {
    <xnm:warning> {
      call jcs:edit-path();
      <message> "filter is missing final 'then accept' rule";
    }
    call jcs:emit-change() {
      with $content = {
        <term> {
          <name> "very-last";
          <junos:comment> "This term was added by a commit script";
          <then> {
            <accept>;
          }
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **add-accept.xsl** or **add-accept.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **add-accept.slax**.

```
system {
    scripts {
        commit {
            file add-accept.xsl;
        }
    }
}
firewall {
    policer sgt-friday {
        if-exceeding {
            bandwidth-percent 10;
            burst-size-limit 250k;
        }
        then discard;
    }
    family inet {
        filter test {
            term one {
                from {
                    interface t1-0/0/0;
                }
                then {
                    count ten-network;
                    discard;
                }
            }
            term two {
                from {
                    forwarding-class assured-forwarding;
                }
                then discard;
            }
        }
    }
}
interfaces {
    t1-0/0/0 {
        unit 0 {
            family inet {
                policer output sgt-friday;
                filter input test;
```

```
        }
      }
    }
  }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The script requires that all firewall filters end with an explicit **then accept** statement. The sample configuration stanzas include the **test** filter with two terms but do not include an explicit **then accept** statement. When you issue the **commit** command, the script adds the missing **then accept** statement and commits the configuration. When you issue the **commit** command, the following output appears:

```
[edit]
user@host# commit
[edit firewall family inet filter test]
  warning: filter is missing final 'then accept' rule
commit complete
```

In configuration mode, issue the **show firewall** command to review the modified configuration. The following output appears:

```
[edit]
user@host# show firewall
policer sgt-friday {
  if-exceeding {
    bandwidth-percent 10;
    burst-size-limit 250k;
    }
  then discard;
}
family inet {
  filter test {
```

```
    term one {
      from {
        interface t1-0/0/0;
      }
      then {
        count ten-network;
        discard;
      }
    }
    term two {
      from {
        forwarding-class assured-forwarding;
      }
      then {
        discard;
      }
    }
    term very-last {
      then accept; /* This term was added by a commit script */
    }
  }
}
```

## Example: Adding T1 Interfaces to a RIP Group

This example shows how to use commit scripts to decrease the amount of manual configuration, specifically how to add every T1 interface configured at the **[edit interfaces]** hierarchy level to the **[edit protocols rip group test]** hierarchy level.



### Requirements

This example uses a device running Junos OS with T1 interfaces.

### Overview and Commit Script

If you want to enable RIP on an interface, you must make changes at both the **[edit interfaces]** and **[edit protocols rip]** hierarchy levels. This example shows how to use commit scripts to add every T1 interface configured at the **[edit interfaces]** hierarchy level to the **[edit protocols rip group test]** hierarchy level. This example includes no error, warning, or system log messages. The changes to the configuration are made silently.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
```

```
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="all-t1"
                  select="interfaces/interface[starts-with(name, 't1-')]"/>
    <xsl:if test="$all-t1">
      <change>
        <protocols>
          <rip>
            <group>
              <name>test</name>
              <xsl:for-each select="$all-t1">
                <xsl:variable name="ifname" select="concat(name, '.0')"/>
                <neighbor>
                  <name><xsl:value-of select="$ifname"/></name>
                </neighbor>
              </xsl:for-each>
            </group>
          </rip>
        </protocols>
      </change>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $all-t1 = interfaces/interface[starts-with(name, 't1-')];
  if ($all-t1) {
    <change> {
      <protocols> {
        <rip> {
          <group> {
            <name> "test";
            for-each ($all-t1) {
              var $ifname = name _ '.0';
              <neighbor> {
                <name> $ifname;
              }
            }
          }
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **rip-t1.xsl** or **rip-t1.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **rip-t1.slax**.

```
system {
    scripts {
        commit {
            file rip-t1.xsl;
        }
    }
}
interfaces {
    t1-0/0/0 {
        unit 0 {
            family iso;
        }
    }
    t1-0/0/1 {
        unit 0 {
            family iso;
        }
    }
    t1-0/0/2 {
        unit 0 {
            family iso;
        }
    }
    t1-0/0/3 {
        unit 0 {
            family iso;
        }
    }
    t1-0/1/0 {
        unit 0 {
            family iso;
        }
    }
    t1-0/1/1 {
        unit 0 {
            family iso;
        }
    }
    t1-0/1/2 {
        unit 0 {
            family iso;
        }
    }
```

```
    t1-0/1/3 {
      unit 0 {
        family iso;
      }
    }
  }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Issue the **show protocols rip group test** command. All T1 interfaces should now appear under the **[edit protocols rip group test]** hierarchy level.

```
[edit]
user@host# show protocols rip group test
neighbor t1-0/0/0.0;
neighbor t1-0/0/1.0;
neighbor t1-0/0/2.0;
neighbor t1-0/0/3.0;
neighbor t1-0/1/0.0;
neighbor t1-0/1/1.0;
neighbor t1-0/1/2.0;
neighbor t1-0/1/3.0;
```

## Example: Assigning a Classifier

For each interface configured with the IPv4 protocol family, this commit script automatically assigns a specified classifier, which associates incoming packets with a forwarding class and loss priority as well as assigns packets to an output queue.


- Requirements on page 94
- Overview and Commit Script on page 94

## Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

In the Junos OS class of service (CoS), classifiers allow you to associate incoming packets with a forwarding class and loss priority and, based on the associated forwarding class, assign packets to output queues. After you configure a classifier, you must assign it to an input interface.

For each interface configured with the IPv4 protocol family, this script automatically assigns a specified classifier called **fc-q3**. The **fc-q3** classifier must be configured at the **[edit class-of-service]** hierarchy level.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="cos-all" select="class-of-service"/>
    <xsl:for-each
        select="interfaces/interface[contains(name, '/')]/unit[family/inet]">
      <xsl:variable name="ifname" select="../name"/>
      <xsl:variable name="unit" select="name"/>
      <xsl:variable name="cos"
        select="$cos-all/interfaces[name = $ifname]"/>
      <xsl:if test="not($cos/unit[name = $unit])">
        <xsl:call-template name="jcs:emit-change">
          <xsl:with-param name="message">
            <xsl:text>Adding CoS forwarding class for </xsl:text>
            <xsl:value-of select="concat($ifname, '.', $unit)"/>
          </xsl:with-param>
          <xsl:with-param name="dot" select="$cos-all"/>
          <xsl:with-param name="content">
            <interfaces>
              <name><xsl:value-of select="$ifname"/></name>
              <unit>
                <name><xsl:value-of select="$unit"/></name>
                <forwarding-class>fc-q3</forwarding-class>
              </unit>
            </interfaces>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:if>
    </xsl:for-each>
```

```
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $cos-all = class-of-service;
  for-each (interfaces/interface[contains(name, '/')]/unit[family/inet]) {
    var $ifname = ../name;
    var $unit = name;
    var $cos = $cos-all/interfaces[name = $ifname];
    if (not($cos/unit[name = $unit])) {
      call jcs:emit-change($dot = $cos-all) {
        with $message = {
          expr "Adding CoS forwarding class for ";
          expr $ifname _ '.' _ $unit;
        }
        with $content = {
          <interfaces> {
            <name> $ifname;
            <unit> {
              <name> $unit;
              <forwarding-class> "fc-q3";
            }
          }
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **classifier.xsl** or **classifier.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **classifier.slax**.

```
system {
  scripts {
    commit {
      file classifier.xsl;
    }
  }
}
```

```
interfaces {
   fe-0/0/0 {
      unit 0 {
         family inet {
            address 10.168.16.2/24;
         }
      }
   }
}
class-of-service {
   forwarding-classes {
      queue 3 fc-q3;
   }
   classifiers {
      inet-precedence fc-q3 {
         forwarding-class fc-q3 {
            loss-priority low code-points 010;
         }
      }
   }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. In the test configuration stanzas, the **fe-0/0/0.0** interface is configured with the **family inet** statement. Because the interface is configured with the IPv4 protocol family, the script automatically assigns the **fc-q3** classifier to the interface, which is indicated in the **commit** command output.

```
[edit]
user@host# commit
[edit interfaces interface fe-0/0/0 unit 0]
 warning: Adding CoS forwarding class for fe-0/0/0.0
```

```
commit complete
```

View the configuration to verify that the script-generated changes are present. Issue the **show class-of-service** configuration mode command. The output shows that the **fe-0/0/0.0** interface has been assigned the **fc-q3** classifier:

```
[edit]
user@host# show class-of-service
classifiers {
  inet-precedence fc-q3 {
    forwarding-class fc-q3 {
      loss-priority low code-points 010;
    }
  }
}
forwarding-classes {
  queue 3 fc-q3;
}
interfaces {
  fe-0/0/0 {
    unit 0 {
      forwarding-class fc-q3;  # Added by commit script
    }
  }
}
```

## Example: Automatically Configuring Logical Interfaces and IP Addresses

Every interface you configure requires at least one logical unit and one IP address. Asynchronous Transfer Mode (ATM) interfaces also require a virtual circuit identifier (VCI) for each logical interface. If you need to configure multiple logical units on an interface, you can use a commit script and macro to complete the task quickly and with no errors.



### Requirements

This example uses a device running Junos OS with physical ATM interfaces.

### Overview and Commit Script

The following commit script expands an **apply-macro** statement that provides the name of a physical ATM interface and a set of parameters that specify how to configure a number of logical units on the interface. The units and VCI numbers are numbered sequentially from the **$unit** variable to the **$max** variable and are given IP addresses starting at the **$address** variable. To loop through the logical units, Extensible Stylesheet Language Transformations (XSLT) uses recursion, which is implemented in the

**<emit-interface>** template. Calculation of the next address is performed in the **<next-address>** template.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/apply-macro">
      <xsl:variable name="device" select="name"/>
      <xsl:variable name="address" select="data[name='address']/value"/>
      <xsl:variable name="max" select="data[name='max']/value"/>
      <xsl:variable name="unit" select="data[name='unit']/value"/>
      <xsl:variable name="real-max">
        <xsl:choose>
          <xsl:when test="string-length($max) &gt; 0">
            <xsl:value-of select="$max"/>
          </xsl:when>
          <xsl:otherwise>0</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:variable name="real-unit">
        <xsl:choose>
          <xsl:when test="string-length($unit) &gt; 0">
            <xsl:value-of select="$unit"/>
          </xsl:when>
          <xsl:when test="contains($device, '.')">
            <xsl:value-of select="substring-after($device, '.')"/>
          </xsl:when>
          <xsl:otherwise>0</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:variable name="real-device">
        <xsl:choose>
          <xsl:when test="contains($device, '.')">
            <xsl:value-of select="substring-before($device, '.')"/>
          </xsl:when>
          <xsl:otherwise><xsl:value-of select="$device"/></xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <transient-change>
        <interfaces>
          <interface>
            <name><xsl:value-of select="$real-device"/></name>
            <xsl:call-template name="emit-interface">
              <xsl:with-param name="address" select="$address"/>
              <xsl:with-param name="unit" select="$real-unit"/>
              <xsl:with-param name="max" select="$real-max"/>
            </xsl:call-template>
          </interface>
```

```
        </interfaces>
      </transient-change>
    </xsl:for-each>
  </xsl:template>
  <xsl:template name="emit-interface">
    <xsl:param name="max"/>
    <xsl:param name="unit"/>
    <xsl:param name="address"/>
    <unit>
      <name><xsl:value-of select="$unit"/></name>
      <vci><xsl:value-of select="$unit"/></vci>
      <family>
        <inet>
          <address><xsl:value-of select="$address"/></address>
        </inet>
      </family>
    </unit>
    <xsl:if test="$max &gt; $unit">
      <xsl:call-template name="emit-interface">
        <xsl:with-param name="address">
          <xsl:call-template name="next-address">
            <xsl:with-param name="address" select="$address"/>
          </xsl:call-template>
        </xsl:with-param>
        <xsl:with-param name="unit" select="$unit + 1"/>
        <xsl:with-param name="max" select="$max"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
  <xsl:template name="next-address">
    <xsl:param name="address"/>
    <xsl:variable name="arg-prefix" select="substring-after($address, '/')"/>
    <xsl:variable name="arg-addr" select="substring-before($address, '/')"/>
    <xsl:variable name="addr">
      <xsl:choose>
        <xsl:when test="string-length($arg-addr) &gt; 0">
          <xsl:value-of select="$arg-addr"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$address"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="prefix">
      <xsl:choose>
        <xsl:when test="string-length($arg-prefix) &gt; 0">
          <xsl:value-of select="$arg-prefix"/>
        </xsl:when>
        <xsl:otherwise>32</xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="a1" select="substring-before($addr, '.')"/>
    <xsl:variable name="a234" select="substring-after($addr, '.')"/>
    <xsl:variable name="a2" select="substring-before($a234, '.')"/>
    <xsl:variable name="a34" select="substring-after($a234, '.')"/>
    <xsl:variable name="a3" select="substring-before($a34, '.')"/>
```

```
    <xsl:variable name="a4" select="substring-after($a34, '.')"/>
    <xsl:variable name="r3">
      <xsl:choose>
        <xsl:when test="$a4 &lt; 255">
          <xsl:value-of select="$a3"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$a3 + 1"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="r4">
      <xsl:choose>
        <xsl:when test="$a4 &lt; 255">
          <xsl:value-of select="$a4 + 1"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="0"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:value-of select="$a1"/>
    <xsl:text>.</xsl:text>
    <xsl:value-of select="$a2"/>
    <xsl:text>.</xsl:text>
    <xsl:value-of select="$r3"/>
    <xsl:text>.</xsl:text>
    <xsl:value-of select="$r4"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="$prefix"/>
  </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/apply-macro) {
    var $device = name;
    var $address = data[name='address']/value;
    var $max = data[name='max']/value;
    var $unit = data[name='unit']/value;
    var $real-max = {
      if (string-length($max) > 0) {
        expr $max;
      } else {
        expr "0";
      }
    }
    var $real-unit = {
      if (string-length($unit) > 0) {
        expr $unit;
      } else if (contains($device, '.')) {
```

```
            expr substring-after($device, '.');
          } else {
            expr "0";
          }
        }
        var $real-device = {
          if (contains($device, '.')) {
            expr substring-before($device, '.');
          } else {
            expr $device;
          }
        }
        <transient-change> {
          <interfaces> {
            <interface> {
              <name> $real-device;
              call emit-interface($address, $unit = $real-unit, $max = $real-max);
            }
          }
        }
      }
    }
    emit-interface ($max, $unit, $address) {
      <unit> {
        <name> $unit;
        <vci> $unit;
        <family> {
          <inet> {
            <address> $address;
          }
        }
      }
      if ($max > $unit) {
        call emit-interface($unit = $unit + 1, $max) {
          with $address = {
            call next-address($address);
          }
        }
      }
    }
    next-address ($address) {
      var $arg-prefix = substring-after($address, '/');
      var $arg-addr = substring-before($address, '/');
      var $addr = {
        if (string-length($arg-addr) > 0) {
          expr $arg-addr;
        } else {
          expr $address;
        }
      }
      var $prefix = {
        if (string-length($arg-prefix) > 0) {
          expr $arg-prefix;
        } else {
          expr "32";
        }
```

```
    }
    var $a1 = substring-before($addr, '.');
    var $a234 = substring-after($addr, '.');
    var $a2 = substring-before($a234, '.');
    var $a34 = substring-after($a234, '.');
    var $a3 = substring-before($a34, '.');
    var $a4 = substring-after($a34, '.');
    var $r3 = {
      if ($a4 < 255) {
        expr $a3;
      } else {
        expr $a3 + 1;
      }
    }
    var $r4 = {
      if ($a4 < 255) {
        expr $a4 + 1;
      } else {
        expr 0;
      }
    }
    expr $a1;
    expr ".";
    expr $a2;
    expr ".";
    expr $r3;
    expr ".";
    expr $r4;
    expr "/";
    expr $prefix;
  }
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and run the script:

1. Copy the XSLT or SLAX script into a text file, name the file **atm-logical.xsl** or **atm-logical.slax** as appropriate, and download it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **atm-logical.slax**.

```
system {
  scripts {
    commit {
      allow-transients;
      file atm-logical.xsl;
    }
  }
}
interfaces {
  apply-macro at-1/2/3 {
```

```
        address 10.12.13.14/20;
        max 200;
        unit 32;
      }
      at-1/2/3 {
        atm-options {
          pic-type atm2;
          vpi 0;
        }
      }
    }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the correct changes are integrated into the configuration.

**Action** Before you commit the configuration, you can verify that the commit script will produce the correct results by issuing the **show interfaces at-1/2/3 | display commit-scripts** configuration mode command. After you commit the configuration, you can review the active configuration by issuing the **show configuration interfaces at-1/2/3** operational mode command. The following output appears:

```
atm-options {
  pic-type atm2;
  vpi 0;
}
unit 32 {
  vci 32;
  family inet {
    address 10.12.13.14/20;
  }
}
unit 33 {
  vci 33;
  family inet {
```

```
        address 10.12.13.15/20;
      }
    }
    unit 34 {
      vci 34;
      family inet {
        address 10.12.13.16/20;
      }
    }
    unit 35 {
      vci 35;
      family inet {
        address 10.12.13.17/20;
      }
    }
    ... Logical units 36 through 199 are omitted for brevity ...
    unit 200 {
      vci 200 ;
      family inet {
        address 10.12.13.182/20;
      }
    }
```

**Meaning** The **| display commit-scripts** option displays the configuration data after all commit scripts have been applied. The output includes both persistent and transient changes. If the appropriate **unit** and **vci** are configured on each ATM interface, the commit script executes successfully during a commit operation. After you commit the configuration, you can review the active configuration by issuing the **show configuration interfaces at-1/2/3** operational mode command.

## Example: Configuring Administrative Groups for LSPs

Administrative groups, also known as link coloring or resource classes, are manually assigned attributes that describe the color of links. Links with the same color conceptually belong to the same class. You can use administrative groups to implement a variety of policy-based label-switched path (LSP) setups.

This commit script example searches for **apply-macro** statements with the **color** parameter included at the **[edit protocols mpls]** hierarchy level. For each **apply-macro** statement, the script uses the data provided to generate a transient change and expand the macro into a standard Junos OS administrative group for LSPs.



### Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

In this example, the Junos OS management process (mgd) inspects the configuration, looking for **apply-macro** statements. For each **apply-macro** statement with the **color** parameter included at the **[edit protocols mpls]** hierarchy level, the script generates a transient change, using the data provided within the **apply-macro** statement to expand the macro into a standard Junos OS administrative group for LSPs.

For this example to work, an **apply-macro** statement must be included at the **[edit protocols mpls]** hierarchy level with a set of addresses, a **color** parameter, and a **group-value** parameter. The commit script converts each address to an LSP configuration and converts the **color** parameter into an administrative group.

For a line-by-line explanation of this script, see "Example: Creating Custom Configuration Syntax with Macros" on page 77.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="mpls" select="protocols/mpls"/>
    <xsl:for-each select="$mpls/apply-macro[data/name = 'color']">
      <xsl:variable name="color" select="data[name = 'color']/value"/>
      <xsl:for-each select="$mpls/apply-macro[data/name = 'group-value']">
        <xsl:variable name="group-value" select="data[name =
            'group-value']/value"/>
        <transient-change>
          <protocols>
            <mpls>
              <admin-groups>
                <name>
                  <xsl:value-of select="$color"/>
                </name>
                <group-value>
                  <xsl:value-of select="$group-value"/>
                </group-value>
              </admin-groups>
              <xsl:for-each select="data[not(value)]/name">
                <label-switched-path>
                  <name>
                    <xsl:value-of select="concat($color, '-lsp-', .)"/>
                  </name>
                  <to><xsl:value-of select="."/></to>
                  <admin-group>
                    <include-any>
                      <xsl:value-of select="$color"/>
                    </include-any>
```

```
                    </admin-group>
                  </label-switched-path>
                </xsl:for-each>
              </mpls>
            </protocols>
          </transient-change>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $mpls = protocols/mpls;
  for-each ($mpls/apply-macro[data/name = 'color']) {
    var $color = data[name = 'color']/value;
    for-each ($mpls/apply-macro[data/name = 'group-value']) {
      var $group-value = data[name = 'group-value']/value;
      <transient-change> {
        <protocols> {
          <mpls> {
            <admin-groups> {
              <name> $color;
              <group-value> $group-value;
            }
            for-each (data[not(value)]/name) {
              <label-switched-path> {
                <name> $color _ '-lsp-' _ .;
                <to> .;
                <admin-group> {
                  <include-any> $color;
                }
              }
            }
          }
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **lsp-admin.xsl** or **lsp-admin.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **lsp-admin.slax**.

   ```
   system {
     scripts {
       commit {
         allow-transients;
         file lsp-admin.xsl;
       }
     }
   }
   protocols {
     mpls {
       apply-macro blue-type-lsp {
         10.1.1.1;
         10.2.2.2;
         10.3.3.3;
         10.4.4.4;
         color blue;
         group-value 0;
       }
     }
   }
   ```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Issue the **show protocols mpls | display commit-scripts** configuration mode command and review the output. Adding the **| display commit-scripts** option allows you to see the configuration statements that are generated by transient changes.

**With Script-Generated Changes** When you issue the **show protocols mpls | display commit-scripts** configuration mode command, the following output appears:

```
[edit]
user@host# show protocols mpls | display commit-scripts
apply-macro blue-type-lsp {
   10.1.1.1;
   10.2.2.2;
   10.3.3.3;
   10.4.4.4;
   color blue;
   group-value 0;
}
admin-groups {
   blue 0;
}
label-switched-path blue-lsp-10.1.1.1 {
   to 10.1.1.1;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.2.2.2 {
   to 10.2.2.2;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.3.3.3 {
   to 10.3.3.3;
   admin-group include-any blue;
}
label-switched-path blue-lsp-10.4.4.4 {
   to 10.4.4.4;
   admin-group include-any blue;
}
```

**Without Script-Generated Changes** The output of the **show protocols mpls | display commit-scripts no-transients** configuration mode command excludes the **label-switched-path** statements:

```
[edit]
user@host# show protocols mpls | display commit-scripts no-transients
apply-macro blue-type-lsp {
   10.1.1.1;
   10.2.2.2;
   10.3.3.3;
   10.4.4.4;
   color blue;
   group-value 0;
```

```
}
```

When you issue the **show protocols mpls** command without the piped **display commit-scripts no-transients** command, you see the same output because this script does not generate any persistent changes:

```
[edit]
user@host# show protocols mpls
apply-macro blue-type-lsp {
   10.1.1.1;
   10.2.2.2;
   10.3.3.3;
   10.4.4.4;
   color blue;
   group-value 0;
}
```

## Example: Configuring a Default Encapsulation Type

This commit script example configures default Cisco HDLC encapsulation on SONET/SDH interfaces not configured as aggregate interfaces.



### Requirements

This example uses a device running Junos OS with SONET/SDH interfaces.

### Overview and Commit Script

Point-to-Point Protocol (PPP) encapsulation is the default encapsulation type for physical interfaces. You do not need to configure encapsulation for any physical interfaces that support PPP encapsulation. If you do not configure encapsulation, PPP is used by default. For physical interfaces that do not support PPP encapsulation, you must configure an encapsulation to use for packets transmitted on the interface.

This example configures default Cisco HDLC encapsulation on SONET/SDH interfaces not configured as aggregate interfaces. The **$tag** variable is passed to the **jcs:emit-change** template as **transient-change**, so this change is not copied to the candidate configuration.

Simply including configuration groups in the configuration does not enable you to test whether the **aggregate** statement is included for an interface at the **[edit interfaces *interface-name* sonet-options]** hierarchy level. A commit script can perform this test and set the encapsulation only on nonaggregated interfaces. The script written to perform this test has the following syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')
                and not(sonet-options/aggregate)]">
      <xsl:call-template name="jcs:emit-change">
        <xsl:with-param name="tag" select="'transient-change'"/>
        <xsl:with-param name="content">
          <encapsulation>cisco-hdlc</encapsulation>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/interface[starts-with(name, 'so-') and
            not(sonet-options/aggregate)]) {
    call jcs:emit-change($tag = 'transient-change') {
      with $content = {
        <encapsulation> "cisco-hdlc";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **so-encap.xsl** or **so-encap.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **so-encap.slax**.

```
system {
  scripts {
    commit {
      allow-transients;
      file so-encap.xsl;
    }
  }
```

```
}
interfaces {
   so-1/2/2 {
      sonet-options {
         aggregate as0;
      }
   }
   so-1/2/3 {
      unit 0 {
         family inet {
            address 10.0.0.3/32;
         }
      }
   }
   so-1/2/4 {
      unit 0 {
         family inet {
            address 10.0.0.4/32;
         }
      }
   }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** When you issue the **commit** command, the commit script tests for SONET/SDH interfaces that are not configured as aggregate interfaces and sets the default encapsulation type on the nonaggregated interfaces to Cisco HDLC encapsulation. This is implemented as a **transient-change**. Even though the transient changes are in effect, they are not, by default, displayed in the normal output of the **show interfaces** command.

```
[edit]
```

```
user@host# show interfaces
so-1/2/2 {
    sonet-options {
        aggregate as0;
    }
}
so-1/2/3 {
    unit 0 {
        family inet {
            address 10.0.0.3/32;
        }
    }
}
so-1/2/4 {
    unit 0 {
        family inet {
            address 10.0.0.4/32;
        }
    }
}
```

To view the configuration with the transient changes, issue the **show interfaces | display commit-scripts** command:

```
[edit]
user@host# show interfaces | display commit-scripts
so-1/2/2 {
    sonet-options {  # The presence of these statements prevents the
        aggregate as0;  # transient change from affecting this interface.
    }
}
so-1/2/3 {
    encapsulation cisco-hdlc;  # Added by transient change.
    unit 0 {
        family inet {
            address 10.0.0.3/32;
        }
    }
}
so-1/2/4 {
    encapsulation cisco-hdlc;  # Added by transient change.
    unit 0 {
        family inet {
            address 10.0.0.4/32;
        }
    }
}
```

## Example: Configuring Dual Routing Engines

If your device has redundant (also called *dual*) Routing Engines, your Junos OS configuration can be complex. This example shows how you can use commit scripts to simplify and control the configuration of dual Routing Engine platforms.



### Requirements

This example uses a device running Junos OS with dual Routing Engines.

### Overview and Commit Script

Junos OS supports two special configuration groups: **re0** and **re1**. When these groups are applied using the **apply-groups [ re0 re1 ]** statement, they take effect if the Routing Engine name matches the group name. Statements included at the **[edit groups re0]** hierarchy level are inherited only on the Routing Engine named RE0, and statements included at the **[edit groups re1]** hierarchy level are inherited only on the Routing Engine named RE1.

This example includes two commit scripts. The first script, **dual-re.xsl**, generates a warning if the **system host-name** statement, any IP version 4 (IPv4) interface address, or the **fxp0** interface configuration is configured in the target configuration instead of in a configuration group.

The second script, **dual-re2.xsl**, first checks whether the hostname configuration is configured and then checks whether it is configured in a configuration group. The **otherwise** construct generates an error message if the hostname is not configured at all. The first **when** construct allows the script to do nothing if the hostname is already configured in a configuration group. The second **when** construct takes effect when the hostname is configured in the target configuration. In this case, the script generates a transient change that places the hostname configuration into the **re0** and **re1** configuration groups, copies the configured hostname into those groups, concatenates each group hostname with **-RE0** and **-RE1**, and deactivates the hostname in the target configuration so the configuration group hostnames can be inherited.

The example scripts are shown in both XSLT and SLAX syntax:

**XSLT Syntax: dual-re.xsl Script**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="system/host-name |
```

```
                interfaces/interface/unit/family/inet/address |
                interfaces/interface[name = 'fxp0']">
          <xsl:if test="not(@junos:group) or not(starts-with(@junos:group, 're'))">
            <xnm:warning>
              <xsl:call-template name="jcs:edit-path">
                <xsl:with-param name="dot" select=".."/>
              </xsl:call-template>
              <xsl:call-template name="jcs:statement"/>
              <message>
                <xsl:text>statement should not be in target</xsl:text>
                <xsl:text> configuration on dual RE system</xsl:text>
              </message>
            </xnm:warning>
          </xsl:if>
        </xsl:for-each>
      </xsl:template>
    </xsl:stylesheet>
```

**XSLT Syntax: dual-re2.xsl Script**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="hn" select="system/host-name"/>
    <xsl:choose>
      <xsl:when test="$hn/@junos:group"/>
      <xsl:when test="$hn">
        <transient-change>
          <groups>
            <name>re0</name>
            <system>
              <host-name>
                <xsl:value-of select="concat($hn, '-RE0')"/>
              </host-name>
            </system>
          </groups>
          <groups>
            <name>re1</name>
            <system>
              <host-name>
                <xsl:value-of select="concat($hn, '-RE1')"/>
              </host-name>
            </system>
          </groups>
          <system>
            <host-name inactive="inactive"/>
          </system>
        </transient-change>
      </xsl:when>
      <xsl:otherwise>
        <xnm:error>
          <message>Missing [system host-name]</message>
```

```
        </xnm:error>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax: dual-re.xsl Script**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (system/host-name | interfaces/interface/unit/family/inet/address |
                interfaces/interface[name = 'fxp0']) {
    if (not(@junos:group) or not(starts-with(@junos:group, 're'))) {
      <xnm:warning> {
        call jcs:edit-path($dot = ..);
        call jcs:statement();
        <message> {
          expr "statement should not be in target";
          expr " configuration on dual RE system";
        }
      }
    }
  }
}
```

**SLAX Syntax: dual-re2.xsl Script**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $hn = system/host-name;
  if ($hn/@junos:group) {
  }
  else if ($hn) {
    <transient-change> {
      <groups> {
        <name> "re0";
        <system> {
          <host-name> $hn _ '-RE0';
        }
      }
      <groups> {
        <name> "re1";
        <system> {
          <host-name> $hn _ '-RE1';
        }
      }
      <system> {
        <host-name inactive="inactive">;
      }
```

```
    }
    else {
      <xnm:error> {
        <message> "Missing [system host-name]";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and run the scripts:

1. Copy the XSLT or SLAX scripts into two text files, name the files **dual-re.xsl** and **dual-re2.xsl** or **dual-re.slax** and **dual-re2.slax** as appropriate, and copy them to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filenames at the **[edit system scripts commit file]** hierarchy level to **dual-re.slax** and **dual-re2.slax**.

```
groups {
  re0 {
    interfaces {
      fxp0 {
        unit 0 {
          family inet {
            address 10.0.0.1/24;
          }
        }
      }
    }
  }
}
apply-groups re0;
system {
  host-name router1;
  scripts {
    commit {
      file dual-re.xsl;
      file dual-re2.xsl;
    }
  }
}
interfaces {
  fe-0/0/0 {
    unit 0 {
      family inet {
        address 192.168.220.1/30;
      }
    }
  }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Commit Script Changes

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. After the commit operation completes, the device hostname is changed to **router1-RE0**.

```
[edit]
user@host# commit
[edit system]
  'host-name router1;'
  warning: statement should not be in target configuration on dual RE system
[edit interfaces interface fe-0/0/0 unit 0 family inet]
  'address 192.168.220.1/30;'
  warning: statement should not be in target configuration on dual RE system
commit complete
```

# Example: Configuring an Interior Gateway Protocol on an Interface

This commit script example uses a macro to automatically include an interface at the **[edit protocols]** hierarchy level and to configure the proper interior gateway protocol (IGP) on the interface.



## Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

When you add a new interface to an OSPF or IS-IS domain, you must configure the interface at multiple hierarchy levels, including **[edit interfaces]** and **[edit protocols]**. This example uses a commit script and macro to automatically include the interface at the **[edit protocols]** hierarchy level and to configure the proper IGP on the interface, either OSPF or IS-IS, depending on the content of an **apply-macro** statement that you include in the interface configuration. This macro allows you to perform more configuration tasks at a single hierarchy level.

In this example, the Junos OS management (mgd) process inspects the configuration, looking for **apply-macro** statements. For each **apply-macro ifclass** statement included at the **[edit interfaces** *interface-name* **unit** *logical-unit-number***]** hierarchy level, the script tests whether the **role** parameter is defined as **cpe**. If so, the script checks the **igp** parameter.

If the **igp** parameter is defined as **isis**, the script includes the relevant interface name at the **[edit protocols isis interface]** hierarchy level.

If the **igp** parameter is defined as **ospf**, the script includes the relevant interface name at the **[edit protocols ospf area** *address* **interface]** hierarchy level. For OSPF, the script references the **area** parameter to determine the correct subnet address of the area.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each
          select="interfaces/interface/unit/apply-macro[name = 'ifclass']">
      <xsl:variable name="role" select="data[name='role']/value"/>
      <xsl:variable name="igp" select="data[name='igp']/value"/>
      <xsl:variable name="ifname">
        <xsl:value-of select="../../name"/>
        <xsl:text>.</xsl:text>
        <xsl:value-of select="../name"/>
      </xsl:variable>
      <xsl:choose>
        <xsl:when test="$role = 'cpe'">
          <change>
            <xsl:choose>
              <xsl:when test="$igp = 'isis'">
                <protocols>
                  <isis>
                    <interface>
                      <name><xsl:value-of select="$ifname"/></name>
                    </interface>
```

```
                    </isis>
                  </protocols>
                </xsl:when>
                <xsl:when test="$igp = 'ospf'">
                  <protocols>
                    <ospf>
                      <area>
                        <name>
                          <xsl:value-of select="data[name='area']/value"/>
                        </name>
                        <interface>
                          <name><xsl:value-of select="$ifname"/></name>
                        </interface>
                      </area>
                    </ospf>
                  </protocols>
                </xsl:when>
              </xsl:choose>
            </change>
          </xsl:when>
        </xsl:choose>
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/interface/unit/apply-macro[name = 'ifclass']) {
    var $role = data[name='role']/value;
    var $igp = data[name='igp']/value;
    var $ifname = {
      expr ../../name;
      expr ".";
      expr ../name;
    }
    if ($role = 'cpe') {
      <change> {
        if ($igp = 'isis') {
          <protocols> {
            <isis> {
              <interface> {
                <name> $ifname;
              }
            }
          }
        }
        else if ($igp = 'ospf') {
          <protocols> {
            <ospf> {
              <area> {
                <name> data[name='area']/value;
```

```
                    <interface> {
                      <name> $ifname;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **if-class.xsl** or **if-class.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **if-class.slax**.

```
system {
  scripts {
    commit {
      file if-class.xsl;
    }
  }
}
interfaces {
  so-1/2/3 {
    unit 0 {
      apply-macro ifclass {
        area 10.4.0.0;
        igp ospf;
        role cpe;
      }
    }
  }
  t3-0/0/0 {
    unit 0 {
      apply-macro ifclass {
        igp isis;
        role cpe;
      }
    }
  }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
```

```
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** View the configuration to verify that the manual changes and the script-generated changes are present.

When you issue the **show interfaces** configuration mode command, the changes added by the sample configuration stanzas should be present in the configuration.

```
[edit]
user@host# show interfaces
t3-0/0/0 {
    unit 0 {
        apply-macro ifclass {
            igp isis;
            role cpe;
        }
    }
}
so-1/2/3 {
    unit 0 {
        apply-macro ifclass {
            area 10.4.0.0;
            igp ospf;
            role cpe;
        }
    }
}
```

When you issue the **show protocols** configuration mode command, the script-generated changes should be present in the configuration.

```
[edit]
user@host# show protocols
isis {
    interface t3-0/0/0.0;
}
ospf {
    area 10.4.0.0 {
```

```
        interface so-1/2/3.0;
    }
}
```

## Example: Controlling IS-IS and MPLS Interfaces

This example shows how to use commit scripts to decrease the amount of manual configuration.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

If you want to enable MPLS on an interface, you must make changes at both the **[edit interfaces]** and **[edit protocols mpls]** hierarchy levels. This example shows how to use commit scripts to decrease the amount of manual configuration.

This example performs two related tasks. If an interface has **[family iso]** configured but not **[family mpls]**, a configuration change is made (using the **jcs:emit-change** template) to enable MPLS. MPLS is not valid on loopback interfaces (**lo*X***), so this script ignores loopback interfaces. Secondly, if the interface is not configured at the **[edit protocols mpls]** hierarchy level, a change is made to add the interface. Both changes are accompanied by appropriate warning messages.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="mpls" select="protocols/mpls"/>
    <xsl:for-each select="interfaces/interface[not(starts-with(name,'lo'))]
                          /unit[family/iso]">
      <xsl:variable name="ifname" select="concat(../name, '.', name)"/>
      <xsl:if test="not(family/mpls)">
        <xsl:call-template name="jcs:emit-change">
          <xsl:with-param name="message">
            <xsl:text>Adding 'family mpls' to ISO-enabled interface</xsl:text>
          </xsl:with-param>
          <xsl:with-param name="content">
            <family>
```

```
                <mpls/>
              </family>
            </xsl:with-param>
          </xsl:call-template>
        </xsl:if>
        <xsl:if test="$mpls and not($mpls/interface[name = $ifname])">
          <xsl:call-template name="jcs:emit-change">
            <xsl:with-param name="message">
              <xsl:text>Adding ISO-enabled interface </xsl:text>
              <xsl:value-of select="$ifname"/>
              <xsl:text> to [protocols mpls]</xsl:text>
            </xsl:with-param>
            <xsl:with-param name="dot" select="$mpls"/>
            <xsl:with-param name="content">
              <interface>
                <name>
                  <xsl:value-of select="$ifname"/>
                </name>
              </interface>
            </xsl:with-param>
          </xsl:call-template>
        </xsl:if>
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $mpls = protocols/mpls;
  for-each (interfaces/interface[not(starts-with(name, "lo"))]/unit[family/iso]) {
    var $ifname = ../name _ '.' _ name;
    if (not(family/mpls)) {
      call jcs:emit-change() {
        with $message = {
          expr "Adding 'family mpls' to ISO-enabled interface";
        }
        with $content = {
          <family> {
            <mpls>;
          }
        }
      }
    }
    if ($mpls and not($mpls/interface[name = $ifname])) {
      call jcs:emit-change($dot = $mpls) {
        with $message = {
          expr "Adding ISO-enabled interface ";
          expr $ifname;
          expr " to [protocols mpls]";
        }
        with $content = {
```

```
            <interface> {
              <name> $ifname;
            }
          }
        }
      }
    }
  }
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **iso.xsl** or **iso.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **iso.slax**.

   ```
   system {
     scripts {
       commit {
         file iso.xsl;
       }
     }
   }
   interfaces {
     lo0 {
       unit 0 {
         family iso;
       }
     }
     so-1/2/3 {
       unit 0 {
         family iso;
       }
     }
     so-1/3/2 {
       unit 0 {
         family iso;
       }
     }
   }
   protocols {
     mpls {
       enable;
     }
   }
   ```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   ```

```
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command.

```
[edit]
user@host# commit
[edit interfaces interface so-1/2/3 unit 0]
  warning: Adding 'family mpls' to ISO-enabled interface
[edit interfaces interface so-1/2/3 unit 0]
  warning: Adding ISO-enabled interface so-1/2/3.0 to [protocols mpls]
[edit interfaces interface so-1/3/2 unit 0]
  warning: Adding 'family mpls' to ISO-enabled interface
[edit interfaces interface so-1/3/2 unit 0]
  warning: Adding ISO-enabled interface so-1/3/2.0 to [protocols mpls]
commit complete
```

Issue the **show interfaces** command. Confirm that the loopback interface is not altered and that the SONET/SDH interfaces are altered.

```
[edit]
user@host# show interfaces
so-1/2/3 {
  unit 0 {
    family iso;
    family mpls;
  }
}
so-1/3/2 {
  unit 0 {
    family iso;
    family mpls;
  }
}
lo0 {
  unit 0 {
    family iso;
  }
}
```

## Example: Controlling LDP Configuration

This commit script example generates a warning on LDP-enabled devices for any interfaces that are configured at either the **[edit protocols ospf]** or **[edit protocols isis]** hierarchy level but are not configured at the **[edit protocols ldp]** hierarchy level. A second test ensures that all LDP-enabled interfaces are configured for an interior gateway protocol (IGP). The example also provides instructions for excluding a particular interface from the commit script LDP test.



### Requirements

This example uses a router running Junos OS.

### Overview and Commit Script

If you want to enable LDP on an interface, you must configure the interface at both the **[edit protocols *routing-protocol-name*]** and **[edit protocols ldp]** hierarchy levels. This example shows how to use commit scripts to ensure that the interface is configured at both levels.

This example tests for interfaces that are configured at either the **[edit protocols ospf]** or **[edit protocols isis]** hierarchy level but not at the **[edit protocols ldp]** hierarchy level. If LDP is not enabled on the device, there is no problem. Otherwise, a warning is generated with the message that the interface does not have LDP enabled.

In case you want some interfaces to be exempt from the LDP test, this script allows you to tag those interfaces as not requiring LDP by including the **apply-macro no-ldp** statement at the **[edit protocols isis interface *interface-name*]** or **[edit protocols ospf area *area-id* interface *interface-name*]** hierarchy level. For example:

```
[edit]
protocols {
   isis {
      interface so-0/1/2.0 {
         apply-macro no-ldp;
      }
   }
}
```

If the **apply-macro no-ldp** statement is included, the warning is not generated.

A second test ensures that all LDP-enabled interfaces are configured for an interior gateway protocol (IGP). As for LDP, you can exempt some interfaces from the test by including the **apply-macro no-igp** statement at the **[edit protocols ldp interface *interface-name*]** hierarchy level. If that statement is not included and no IGP is configured, a warning is generated.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="ldp" select="protocols/ldp"/>
    <xsl:variable name="isis" select="protocols/isis"/>
    <xsl:variable name="ospf" select="protocols/ospf"/>
    <xsl:if test="$ldp">
      <xsl:for-each select="$isis/interface/name |
                    $ospf/area/interface/name">
        <xsl:variable name="ifname" select="."/>
        <xsl:if test="not(../apply-macro[name = 'no-ldp'])
                    and not($ldp/interface[name = $ifname])">
          <xnm:warning>
            <xsl:call-template name="jcs:edit-path"/>
            <xsl:call-template name="jcs:statement"/>
            <message>ldp not enabled for this interface</message>
          </xnm:warning>
        </xsl:if>
      </xsl:for-each>
      <xsl:for-each select="protocols/ldp/interface/name">
        <xsl:variable name="ifname" select="."/>
        <xsl:if test="not(apply-macro[name = 'no-igp'])
                    and not($isis/interface[name = $ifname])
                    and not($ospf/area/interface[name = $ifname])">
          <xnm:warning>
            <xsl:call-template name="jcs:edit-path"/>
            <xsl:call-template name="jcs:statement"/>
            <message>
              <xsl:text>ldp-enabled interface does not have </xsl:text>
              <xsl:text>an IGP configured</xsl:text>
            </message>
          </xnm:warning>
        </xsl:if>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

apply-macro no-ldp;
match configuration {
  var $ldp = protocols/ldp;
  var $isis = protocols/isis;
```

```
var $ospf = protocols/ospf;
if ($ldp) {
  for-each ($isis/interface/name | $ospf/area/interface/name) {
    var $ifname = .;
    if (not(../apply-macro[name = 'no-ldp']) and not($ldp/interface[name =
          $ifname])) {
      <xnm:warning> {
        call jcs:edit-path();
        call jcs:statement();
        <message> "ldp not enabled for this interface";
      }
    }
  }
  for-each (protocols/ldp/interface/name) {
    var $ifname = .;
    if (not(apply-macro[name = 'no-igp']) and not($isis/interface[name =
          $ifname]) and not($ospf/area/interface[name = $ifname])) {
      <xnm:warning> {
        call jcs:edit-path();
        call jcs:statement();
        <message> {
          expr "ldp-enabled interface does not have ";
          expr "an IGP configured";
        }
      }
    }
  }
 }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **ldp.xsl** or **ldp.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **ldp.slax**.

```
system {
  scripts {
    commit {
      file ldp.xsl;
    }
  }
}
protocols {
  isis {
    interface so-1/2/2.0 {
      apply-macro no-ldp;
    }
    interface so-1/2/3.0;
```

```
    }
    ospf {
      area 10.4.0.0 {
        interface ge-3/2/1.0;
        interface ge-2/2/1.0;
      }
    }
    ldp {
      interface ge-1/2/1.0;
      interface ge-2/2/1.0;
    }
  }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration stanzas enable LDP on the device and configure the **so-1/2/2** and **so-1/2/3** interfaces at the **[edit protocols isis]** hierarchy level and the **ge-3/2/1** and **ge-2/2/1** interfaces at the **[edit protocols ospf]** hierarchy level.

Because **ge-2/2/1** is also configured at the **[edit protocols ldp]** hierarchy level, the script does not issue a warning message for this interface during the commit operation. The configuration includes the **apply-macro no-ldp** statement under the **so-1/2/2** interface, so the script does not test this interface or issue a warning message for it, even though it is not configured at the **[edit protocols ldp]** hierarchy.

Neither **so-1/2/3** nor **ge-3/2/1** is configured at the **[edit protocols ldp]** hierarchy level as required by the commit script, so a warning is issued for both interfaces. The **ge-1/2/1** interface is configured at the **[edit protocols ldp]** hierarchy. However, it is not configured for an IGP, so the commit script also issues a warning for the **ge-1/2/1** interface.

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

```
[edit protocols ospf area 10.4.0.0 interface so-1/2/3.0]
    'interface so-1/2/3.0;'
        warning: LDP not enabled for this interface
[edit protocols ospf area 10.4.0.0 interface ge-3/2/1.0]
    'interface ge-3/2/1.0;'
        warning: LDP not enabled for this interface
[edit protocols ldp interface ge-1/2/1.0]
    'interface ge-1/2/1.0;'
        warning: LDP-enabled interface does not have an IGP configured
commit complete
```

## Example: Creating a Complex Configuration Based on a Simple Interface Configuration

This commit script example uses a macro to automatically expand a simple interface configuration.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

This example uses a commit script and macro to automatically expand a simple interface configuration by generating a transient change that assigns a default encapsulation type, configures multiple routing protocols on the interface, and applies multiple configuration groups. The Junos OS management (mgd) process inspects the configuration, looking for **apply-macro params** statements included at the **[edit interfaces *interface-name*]** hierarchy level.

When the script finds an **apply-macro params** statement, it performs the following actions:

- Applies the **interface-details** configuration group to the interface.
- Includes the value of the **description** parameter at the **[edit interfaces *interface-name* description]** hierarchy level.
- Includes the value of the **encapsulation** parameter at the **[edit interfaces *interface-name* encapsulation]** hierarchy level. If the **encapsulation** parameter is not included in the **apply-macro params** statement, the script sets the encapsulation to **cisco-hdlc** as the default.
- Sets the logical unit number to **0** and tests whether the **inet-address** parameter is included in the **apply-macro params** statement. If it is, the script includes the value of the **inet-address** parameter at the **[edit interfaces *interface-name* unit 0 family inet address]** hierarchy level.
- Includes the interface name at the **[edit protocols rsvp interface]** hierarchy level.

- Includes the **level 1 enable** and **metric** statements at the **[edit protocols isis interface *interface-name*]** hierarchy level.
- Includes the **level 2 enable** and **metric** statements at the **[edit protocols isis interface *interface-name*]** hierarchy level.
- Tests whether the **isis-level-1** or **isis-level-1-metric** parameter is included in the **apply-macro params** statement. If one or both of these parameters are included, the script includes the **level 1** statement at the **[edit protocols isis interface *interface-name*]** hierarchy level. If the **isis-level-1** parameter is included, the script also includes the value of the **isis-level-1** parameter (**enable** or **disable**) at the **[edit protocols isis interface *interface-name* level 1]** hierarchy level. If the **isis-level-1-metric** parameter is included, the script also includes the value of the **isis-level-1-metric** parameter at the **[edit protocols isis interface *interface-name* level 1 metric]** hierarchy level.
- Tests whether the **isis-level-2** or **isis-level-2-metric** parameter is included in the **apply-macro params** statement. If one or both of these parameters are included, the script includes the **level 2** statement at the **[edit protocols isis interface *interface-name*]** hierarchy level. If the **isis-level-2** parameter is included, the script also includes the value of the **isis-level-2** parameter (**enable** or **disable**) at the **[edit protocols isis interface *interface-name* level 2]** hierarchy level. If the **isis-level-2-metric** parameter is included, the script also includes the value of the **isis-level-2-metric** parameter at the **[edit protocols isis interface *interface-name* level 2 metric]** hierarchy level.
- Includes the interface name at the **[edit protocols ldp interface]** hierarchy level.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:variable name="top" select="."/>
    <xsl:for-each select="interfaces/interface/apply-macro[name = 'params']">
      <xsl:variable name="description"
          select="data[name = 'description']/value"/>
      <xsl:variable name="inet-address"
          select="data[name = 'inet-address']/value"/>
      <xsl:variable name="encapsulation"
          select="data[name = 'encapsulation']/value"/>
      <xsl:variable name="isis-level-1"
          select="data[name = 'isis-level-1']/value"/>
      <xsl:variable name="isis-level-1-metric"
          select="data[name = 'isis-level-1-metric']/value"/>
      <xsl:variable name="isis-level-2"
          select="data[name = 'isis-level-2']/value"/>
      <xsl:variable name="isis-level-2-metric"
          select="data[name = 'isis-level-2-metric']/value"/>
      <xsl:variable name="ifname" select="concat(../name, '.0')"/>
      <transient-change>
```

```
<interfaces>
  <interface>
    <name><xsl:value-of select="../name"/></name>
    <apply-groups>
      <name>interface-details</name>
    </apply-groups>
    <xsl:if test="$description">
      <description>
        <xsl:value-of select="$description"/>
      </description>
    </xsl:if>
    <encapsulation>
      <xsl:choose>
        <xsl:when test="string-length($encapsulation) &gt; 0">
          <xsl:value-of select="$encapsulation"/>
        </xsl:when>
        <xsl:otherwise>cisco-hdlc</xsl:otherwise>
      </xsl:choose>
    </encapsulation>
    <unit>
      <name>0</name>
      <xsl:if test="string-length($inet-address) &gt; 0">
        <family>
          <inet>
            <address>
              <xsl:value-of select="$inet-address"/>
            </address>
          </inet>
        </family>
      </xsl:if>
    </unit>
  </interface>
</interfaces>
<protocols>
  <rsvp>
    <interface>
      <name><xsl:value-of select="$ifname"/></name>
    </interface>
  </rsvp>
  <isis>
    <interface>
      <name><xsl:value-of select="$ifname"/></name>
      <xsl:if test="$isis-level-1 or $isis-level-1-metric">
        <level>
          <name>1</name>
          <xsl:if test="$isis-level-1">
            <xsl:element name="{$isis-level-1}"/>
          </xsl:if>
          <xsl:if test="$isis-level-1-metric">
            <metric>
              <xsl:value-of select="$isis-level-1-metric"/>
            </metric>
          </xsl:if>
        </level>
      </xsl:if>
      <xsl:if test="$isis-level-2 or $isis-level-2-metric">
```

```
                    <level>
                      <name>2</name>
                      <xsl:if test="$isis-level-2">
                        <xsl:element name="{$isis-level-2}"/>
                      </xsl:if>
                      <xsl:if test="$isis-level-2-metric">
                        <metric>
                          <xsl:value-of select="$isis-level-2-metric"/>
                        </metric>
                      </xsl:if>
                    </level>
                  </xsl:if>
                </interface>
              </isis>
              <ldp>
                <interface>
                  <name><xsl:value-of select="$ifname"/></name>
                </interface>
              </ldp>
            </protocols>
          </transient-change>
        </xsl:for-each>
      </xsl:template>
    </xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  var $top = .;
  for-each (interfaces/interface/apply-macro[name = 'params']) {
    var $description = data[name = 'description']/value;
    var $inet-address = data[name = 'inet-address']/value;
    var $encapsulation = data[name = 'encapsulation']/value;
    var $isis-level-1 = data[name = 'isis-level-1']/value;
    var $isis-level-1-metric = data[name = 'isis-level-1-metric']/value;
    var $isis-level-2 = data[name = 'isis-level-2']/value;
    var $isis-level-2-metric = data[name = 'isis-level-2-metric']/value;
    var $ifname = ../name _ '.0';
    <transient-change> {
      <interfaces> {
        <interface> {
          <name> ../name;
          <apply-groups> {
            <name> "interface-details";
          }
          if ($description) {
            <description> $description;
          }
          <encapsulation> {
            if (string-length($encapsulation) > 0) {
              expr $encapsulation;
            } else {
```

```
            expr "cisco-hdlc";
          }
        }
        <unit> {
          <name> "0";
          if (string-length($inet-address) > 0) {
            <family> {
              <inet> {
                <address> $inet-address;
              }
            }
          }
        }
      }
    }
    <protocols> {
      <rsvp> {
        <interface> {
          <name> $ifname;
        }
      }
      <isis> {
        <interface> {
          <name> $ifname;
          if ($isis-level-1 or $isis-level-1-metric) {
            <level> {
              <name> "1";
              if ($isis-level-1) {
                <xsl:element name="{$isis-level-1}">;
              }
              if ($isis-level-1-metric) {
                <metric> $isis-level-1-metric;
              }
            }
          }
          if ($isis-level-2 or $isis-level-2-metric) {
            <level> {
              <name> "2";
              if ($isis-level-2) {
                <xsl:element name="{$isis-level-2}">;
              }
              if ($isis-level-2-metric) {
                <metric> $isis-level-2-metric;
              }
            }
          }
        }
      }
      <ldp> {
        <interface> {
          <name> $ifname;
        }
      }
    }
  }
}
```

```
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **if-params.xsl** or **if-params.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **if-params.slax**.

   ```
   system {
     scripts {
       commit {
         allow-transients;
         file if-params.xsl;
       }
     }
   }
   groups {
     interface-details {
       interfaces {
         <so-*/*/*> {
           clocking internal;
         }
       }
     }
   }
   interfaces {
     so-1/2/3 {
       apply-macro params {
         description "Link to Hoverville";
         encapsulation ppp;
         inet-address 10.1.2.3/28;
         isis-level-1 enable;
         isis-level-1-metric 50;
         isis-level-2-metric 85;
       }
     }
   }
   ```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** Issue the **show interfaces | display commit-scripts | display inheritance** configuration mode command. The **| display commit-scripts** option displays all the statements that are in the configuration, including statements that are generated by transient changes. The **| display inheritance** option displays inherited configuration data and information about the source group from which the configuration has been inherited. This option also shows interface ranges configuration data in expanded format and information about the source interface-range from which the configuration has been expanded. You should see the following output:

```
[edit]
user@host# show interfaces | display commit-scripts | display inheritance
so-1/2/3 {
    apply-macro params {
        clocking internal;
        description "Link to Hoverville";
        encapsulation ppp;
        inet-address 10.1.2.3/28;
        isis-level-1 enable;
        isis-level-1-metric 50;
        isis-level-2-metric 85;
    }
    description "Link to Hoverville";
    ##
    ## 'internal' was inherited from group 'interface-details'
    ##
    clocking internal;
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.1.2.3/28;
        }
    }
}
```

Issue the **show protocols | display commit-scripts** configuration mode command. You should see the following output:

```
[edit]
user@host# show protocols | display commit-scripts
rsvp {
    interface so-1/2/3.0;
}
```

```
isis {
  interface so-1/2/3.0 {
    level 1 {
      enable;
      metric 50;
    }
    level 2 metric 85;
  }
}
ldp {
  interface so-1/2/3.0;
}
```

## Example: Imposing a Minimum MTU Setting

The maximum transmission unit (MTU) is the greatest amount of data or packet size (in bytes) that can be transferred in one physical frame on a network. In this example, a commit script tests the MTU of SONET/SDH interfaces. If the MTU is less than a specified minimum value, the commit script reports the error and causes the commit operation to fail.



### Requirements

This example uses a device running Junos OS with SONET/SDH interfaces.

### Overview and Commit Script

This example tests the MTU of SONET/SDH interfaces, reports when the MTU is less than the value of the **$min-mtu** parameter, here set to 2048, and causes the commit operation to fail. The **for** loop selects all SONET/SDH interfaces that start with **so-** and that have an MTU statement that is defined and less than the value of **$min-mtu**. For the selected interfaces, the script generates an error, which includes the location of the interface in the configuration hierarchy and the MTU configured for that interface.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:param name="min-mtu" select="2048"/>
  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')
                and mtu and mtu &lt; $min-mtu]">
```

```
        <xnm:error>
          <xsl:call-template name="jcs:edit-path"/>
          <xsl:call-template name="jcs:statement">
            <xsl:with-param name="dot" select="mtu"/>
          </xsl:call-template>
          <message>
            <xsl:text>SONET interfaces must have a minimum MTU of </xsl:text>
            <xsl:value-of select="$min-mtu"/>
            <xsl:text>.</xsl:text>
          </message>
        </xnm:error>
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

param $min-mtu = 2048;
match configuration {
  for-each (interfaces/interface[starts-with(name, 'so-') and mtu and
            mtu < $min-mtu]) {
    <xnm:error> {
      call jcs:edit-path();
      call jcs:statement($dot = mtu);
      <message> {
        expr "SONET interfaces must have a minimum MTU of ";
        expr $min-mtu;
        expr ".";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **so-mtu.xsl** or **so-mtu.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **so-mtu.slax**.

```
system {
  scripts {
    commit {
      file so-mtu.xsl;
    }
  }
```

```
}
interfaces {
   so-1/2/2 {
      mtu 2048;
   }
   so-1/2/3 {
      mtu 576;
   }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Commit Script Output

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration stanzas configure two SONET/SDH interfaces **so-1/2/2** and **so-1/2/3**. The **so-1/2/3** interface is configured with an MTU of 576, so the script generates an error message, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit interfaces interface so-1/2/3]
  'mtu 576;'
    SONET interfaces must have a minimum MTU of 2048.
error: 1 error reported by commit scripts
error: commit script failure
```

## Example: Limiting the Number of ATM Virtual Circuits

This commit script example limits the number of Asynchronous Transfer Mode (ATM) virtual circuits (VCs) configured on an ATM interface.



### Requirements

This example uses a device running Junos OS with an ATM interface.

### Overview and Commit Script

For each ATM interface, the set of corresponding VCs is selected. The number of those VCs, as determined by the built-in Extensible Stylesheet Language Transformations (XSLT) **count()** function, cannot exceed the limit set by the global variable **$limit**. If there are more ATM VCs than **$limit**, a commit error is generated, and the commit operation fails.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:param name="limit" select="10"/>
  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 'at-')]">
      <xsl:variable name="count" select="count(unit)"/>
      <xsl:if test="$count &gt; $limit">
        <xnm:error>
          <edit-path>[edit interfaces]</edit-path>
          <statement><xsl:value-of select="name"/></statement>
          <message>
            <xsl:text>ATM VC limit exceeded; </xsl:text>
            <xsl:value-of select="$count"/>
            <xsl:text> are configured but only </xsl:text>
            <xsl:value-of select="$limit"/>
            <xsl:text> are allowed.</xsl:text>
          </message>
        </xnm:error>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

param $limit = 10;
match configuration {
  for-each (interfaces/interface[starts-with(name, 'at-')]) {
    var $count = count(unit);
    if ($count > $limit) {
      <xnm:error> {
        <edit-path> "[edit interfaces]";
        <statement> name;
        <message> {
          expr "ATM VC limit exceeded; ";
          expr $count;
          expr " are configured but only ";
          expr $limit;
          expr " are allowed.";
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **atm-vc-limit.xsl** or **atm-vc-limit.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **atm-vc-limit.slax**.

```
system {
  scripts {
    commit {
      file atm-vc-limit.xsl;
    }
  }
}
interfaces {
  at-1/2/3 {
    unit 15 {
      family inet {
        address 10.12.13.15/20;
      }
    }
    unit 16 {
      family inet {
```

```
            address 10.12.13.16/20;
          }
        }
        unit 17 {
          family inet {
            address 10.12.13.17/20;
          }
        }
        unit 18 {
          family inet {
            address 10.12.13.18/20;
          }
        }
        unit 19 {
          family inet {
            address 10.12.13.19/20;
          }
        }
        unit 20 {
          family inet {
            address 10.12.13.20/20;
          }
        }
        unit 21 {
          family inet {
            address 10.12.13.21/20;
          }
        }
        unit 22 {
          family inet {
            address 10.12.13.22/20;
          }
        }
        unit 23 {
          family inet {
            address 10.12.13.23/20;
          }
        }
        unit 24 {
          family inet {
            address 10.12.13.24/20;
          }
        }
        unit 25 {
          family inet {
            address 10.12.13.25/20;
          }
        }
        unit 26 {
          family inet {
            address 10.12.13.26/20;
          }
        }
      }
    }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

```
[edit]
user@host# load merge terminal
[Type ^D at a new line to end input]
... Paste the contents of the clipboard here ...
```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

```
user@host# commit
```

## Verification

### Verifying the Commit Script Output

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration stanzas configure 12 virtual circuits on the ATM interface **atm-1/2/3**. Because the commit script only allows 10 ATM VCs to be configured on any ATM interface, the script generates an error, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit interfaces]
  'at-1/2/3'
  ATM VC limit exceeded; 12 are configured but only 10 are allowed.
error: 1 error reported by commit scripts
error: commit script failure
```

## Example: Limiting the Number of E1 Interfaces

This commit script example limits the number of E1 interfaces configured on a Channelized STM1 Intelligent Queuing (IQ) PIC to avoid contention issues with per-unit-schedulers.



## Requirements

This example uses a device running Junos OS with a Channelized STM1 Intelligent Queuing (IQ) PIC.

## Overview and Commit Script

The following script ensures that there are no more than 16 E1 interfaces configured on a channelized STM1 IQ interface. For each channelized STM1 interface (**cstm1-**), the set of corresponding E1 interfaces is selected. The number of those interfaces, as determined by the built-in Extensible Stylesheet Language Transformations (XSLT) **count()** function, cannot exceed the limit set by the global parameter **$limit**. If there are more E1 interfaces than **$limit**, a commit error is generated, and the commit operation fails.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:param name="limit" select="16"/>
  <xsl:template match="configuration">
    <xsl:variable name="interfaces" select="interfaces"/>
    <xsl:for-each select="$interfaces/interface[starts-with(name, 'cstm1-')]">
      <xsl:variable name="triple" select="substring-after(name, 'cstm1-')"/>
      <xsl:variable name="e1name" select="concat('e1-', $triple)"/>
      <xsl:variable name="count"
         select="count($interfaces/interface[starts-with(name, $e1name)])"/>
      <xsl:if test="$count > $limit">
        <xnm:error>
          <edit-path>[edit interfaces]</edit-path>
          <statement><xsl:value-of select="name"/></statement>
          <message>
            <xsl:text>E1 interface limit exceeded on CSTM1 IQ PIC. </xsl:text>
            <xsl:value-of select="$count"/>
            <xsl:text> E1 interfaces are configured, but only </xsl:text>
            <xsl:value-of select="$limit"/>
            <xsl:text> are allowed.</xsl:text>
          </message>
        </xnm:error>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

param $limit = 16;
match configuration {
  var $interfaces = interfaces;
  for-each ($interfaces/interface[starts-with(name, 'cstm1-')]) {
```

```
    var $triple = substring-after(name, 'cstm1-');
    var $e1name = 'e1-' _ $triple;
    var $count = count($interfaces/interface[starts-with(name, $e1name)]);
    if ($count > $limit) {
      <xnm:error> {
        <edit-path> "[edit interfaces]";
        <statement> name;
        <message> {
          expr "E1 interface limit exceeded on CSTM1 IQ PIC. ";
          expr $count;
          expr " E1 interfaces are configured, but only ";
          expr $limit;
          expr " are allowed.";
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **e1-limit.xsl** or **e1-limit.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **e1-limit.slax**.

```
system {
  scripts {
    commit {
      file e1-limit.xsl;
    }
  }
}
interfaces {
  cau4-0/1/0 {
    partition 1 interface-type ce1;
    partition 2-18 interface-type e1;
  }
  cstm1-0/1/0 {
    no-partition interface-type cau4;
  }
  ce1-0/1/0:1 {
    clocking internal;
    e1-options {
      framing g704;
    }
    partition 1 timeslots 1-4 interface-type ds;
  }
  ds-0/1/0:1:1 {
    no-keepalives;
```

```
    dce;
    encapsulation frame-relay;
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.0/31;
      }
    }
  }
  e1-0/1/0:2 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.2/31;
      }
    }
  }
  e1-0/1/0:3 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.4/31;
      }
    }
  }
  e1-0/1/0:4 {
    no-keepalives;
    per-unit-scheduler;
```

```
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.6/31;
      }
    }
  }
  e1-0/1/0:5 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.8/31;
      }
    }
  }
  e1-0/1/0:6 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.10/31;
      }
    }
```

```
}
e1-0/1/0:7 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.12/31;
        }
    }
}
e1-0/1/0:8 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.14/31;
        }
    }
}
e1-0/1/0:9 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
```

```
        family inet {
            address 10.0.0.16/31;
        }
    }
}
e1-0/1/0:10 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.18/31;
        }
    }
}
e1-0/1/0:11 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.20/31;
        }
    }
}
e1-0/1/0:12 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
```

```
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.22/31;
      }
    }
  }
  e1-0/1/0:13 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.24/31;
      }
    }
  }
  e1-0/1/0:14 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
      framing g704;
    }
    lmi {
      lmi-type ansi;
    }
    unit 100 {
      point-to-point;
      dlci 100;
      family inet {
        address 10.0.0.26/31;
      }
    }
  }
  e1-0/1/0:15 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
```

```
        framing g704;
      }
      lmi {
        lmi-type ansi;
      }
      unit 100 {
        point-to-point;
        dlci 100;
        family inet {
          address 10.0.0.28/31;
        }
      }
    }
    e1-0/1/0:16 {
      no-keepalives;
      per-unit-scheduler;
      dce;
      clocking internal;
      encapsulation frame-relay;
      e1-options {
        framing g704;
      }
      lmi {
        lmi-type ansi;
      }
      unit 100 {
        point-to-point;
        dlci 100;
        family inet {
          address 10.0.0.30/31;
        }
      }
    }
    e1-0/1/0:17 {
      no-keepalives;
      per-unit-scheduler;
      dce;
      clocking internal;
      encapsulation frame-relay;
      e1-options {
        framing g704;
      }
      lmi {
        lmi-type ansi;
      }
      unit 100 {
        point-to-point;
        dlci 100;
        family inet {
          address 10.0.0.32/31;
        }
      }
    }
    e1-0/1/0:18 {
      no-keepalives;
      per-unit-scheduler;
```

```
        dce;
        clocking internal;
        encapsulation frame-relay;
        e1-options {
          framing g704;
        }
        lmi {
          lmi-type ansi;
        }
        unit 100 {
          point-to-point;
          dlci 100;
          family inet {
            address 10.0.0.34/31;
          }
        }
      }
    }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Commit Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration stanzas channelize a **cstm1-0/1/0** interface into 17 **E1** interfaces, so the script generates an error, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit interfaces]
  'cstm1-0/1/0'
  E1 interface limit exceeded on CSTM1 IQ PIC.
  17 E1 interfaces are configured, but only 16 are allowed.
error: 1 error reported by commit scripts
error: commit script failure
```

## Example: Loading a Base Configuration

This commit script example sets up a sample base configuration on a device running Junos OS.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

This script is a macro that sets up a device running Junos OS with a sample base configuration. With minimal manual user input, the script automatically configures:

- A device hostname
- Authentication services
- A superuser login
- System log settings
- Some SNMP settings
- System services, such as FTP and Telnet
- Static routes and a policy to redistribute the static routes
- Configuration groups **re0** and **re1**
- An address for the management Ethernet interface (**fxp0**)
- The loopback interface (**lo0**) with the device ID as the loopback address

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

<xsl:variable name="macro-name" select="'config-system.xsl'"/>
<xsl:template match="configuration">
  <xsl:variable name="rid" select="routing-options/router-id"/>
  <xsl:for-each select="apply-macro[name = 'config-system']">
    <xsl:variable name="hostname" select="data[name =
                'host-name']/value"/>
    <xsl:variable name="fxp0-addr" select="data[name =
```

```
            'mgmt-address']/value"/>
<xsl:variable name="backup-router" select="data[name =
            'backup-router']/value"/>
<xsl:variable name="bkup-rtr">
  <xsl:choose>
    <xsl:when test="$backup-router">
      <xsl:value-of select="$backup-router"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="fxp01" select="substring-before($fxp0-addr,
          '.')"/>
      <xsl:variable name="fxp02"
          select="substring-before(substring-after($fxp0-addr, '.'), '.')"/>
      <xsl:variable name="fxp03"
          select="substring-before(substring-after(substring-after(
                $fxp0-addr, '.'), '.'), '.')"/>
      <xsl:variable name="plen" select="substring-after($fxp0-addr, '/')"/>
      <xsl:choose>
        <xsl:when test="$plen = 22">
          <xsl:value-of select="concat($fxp01, '.', $fxp02, '.', $fxp03 div
                4 * 4 + 3, '.254')"/>
        </xsl:when>
        <xsl:when test="$plen = 24">
          <xsl:value-of select="concat($fxp01, '.', $fxp02, '.', $fxp03,
                '.254')"/>
        </xsl:when>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:choose>
  <xsl:when test="not($rid) or not($hostname) or not($fxp0-addr)">
    <xnm:error>
      <message>
        Must set router ID, host-name and mgmt-address to use this script.
      </message>
    </xnm:error>
  </xsl:when>
  <xsl:otherwise>
    <transient-change>
      <system>
      <!-- Set the following -->
        <domain-name>your-domain.net</domain-name>
        <domain-search>domain.net</domain-search>
        <backup-router>
          <address><xsl:value-of select="$bkup-rtr"/></address>
        </backup-router>
        <time-zone>America/Los_Angeles</time-zone>
        <authentication-order>radius</authentication-order>
        <authentication-order>password</authentication-order>
        <root-authentication>
          <encrypted-password>
            $1$Q3CG88jZ$.qhPUZaHdaIMWF2CvxKTe0
          </encrypted-password>
        </root-authentication>
        <name-server>
```

```
    <name>192.168.5.68</name>
  </name-server>
  <name-server>
    <name>172.17.28.100</name>
  </name-server>
  <radius-server>
    <name>192.168.170.241</name>
    <secret>
      $9$4xoDk5T3n/AHkmTQFCA0BIclKWL7sgaRh-bs4GU
    </secret>
  </radius-server>
  <radius-server>
    <name>192.168.4.240</name>
    <secret>
      $9$TQ/t1IcSrKAt0IRheK8X7VYgaZDm5zNdiqmTn6
    </secret>
  </radius-server>
  <login>
    <class>
      <permissions>all</permissions>
    </class>
    <user>
      <name>johnny</name>
      <uid>928</uid>
      <class>superuser</class>
      <authentication>
        <encrypted-password>
          $1$kPU..$w.4FGRAGanJ8U4Yq6sbj7.
        </encrypted-password>
      </authentication>
    </user>
  </login>
  <services>
    <finger/>
    <ftp/>
    <ssh/>
    <telnet/>
    <xnm-clear-text/>
  </services>
  <syslog>
    <user>
      <name>*</name>
      <contents>
        <name>any</name>
        <emergency/>
      </contents>
    </user>
    <host>
      <name>host1</name>
      <contents>
        <name>any</name>
        <notice/>
      </contents>
      <contents>
        <name>interactive-commands</name>
        <any/>
```

```
    </contents>
  </host>
  <file>
    <name>messages</name>
    <contents>
      <name>any</name>
      <notice/>
    </contents>
    <contents>
      <name>any</name>
      <warning/>
    </contents>
    <contents>
      <name>authorization</name>
      <info/>
    </contents>
    <archive>
      <world-readable/>
    </archive>
  </file>
  <file>
    <name>security</name>
    <contents>
      <name>interactive-commands</name>
      <any/>
    </contents>
    <archive>
      <world-readable/>
    </archive>
  </file>
</syslog>
<processes>
  <routing>
    <undocumented><enable/></undocumented>
  </routing>
  <snmp>
    <undocumented><enable/></undocumented>
  </snmp>
  <ntp>
    <undocumented><enable/></undocumented>
  </ntp>
  <inet-process>
    <undocumented><enable/></undocumented>
  </inet-process>
  <mib-process>
    <undocumented><enable/></undocumented>
  </mib-process>
  <undocumented><management><enable/>
  </undocumented></management>
  <watchdog>
    <enable/>
    </watchdog>
  </processes>
  <ntp>
    <boot-server>domain.net</boot-server>
    <server>
```

```
        <name>domainr.net</name>
      </server>
    </ntp>
  </system>
  <snmp>
    <location>Software lab</location>
    <contact>Michael Landon</contact>
    <interface>fxp0.0</interface>
    <community>
      <name>public</name>
      <authorization>read-only</authorization>
      <clients>
        <name>0.0.0.0/0</name>
        <restrict/>
      </clients>
      <clients>
        <name>192.168.1.252/32</name>
      </clients>
      <clients>
        <name>10.197.169.222/32</name>
      </clients>
      <clients>
        <name>10.197.169.188/32</name>
      </clients>
      <clients>
        <name>10.197.169.193/32</name>
      </clients>
      <clients>
        <name>192.168.65.46/32</name>
      </clients>
      <clients>
        <name>10.209.152.0/23</name>
      </clients>
    </community>
    <community>
      <name>private</name>
      <authorization>read-write</authorization>
      <clients>
        <name>0.0.0.0/0</name>
        <restrict/>
      </clients>
      <clients>
        <name>10.197.169.188/32</name>
      </clients>
    </community>
  </snmp>
  <routing-options>
    <static>
      <junos:comment>/* safety precaution */</junos:comment>
      <route>
        <name>0.0.0.0/0</name>
        <discard/>
        <retain/>
        <no-readvertise/>
      </route>
      <junos:comment>/* corporate net */</junos:comment>
```

```
    <route>
      <name>172.16.0.0/12</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* lab nets */</junos:comment>
    <route>
      <name>192.168.0.0/16</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* reflector */</junos:comment>
    <route>
      <name>10.17.136.192/32</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* another lab1*/</junos:comment>
    <route>
      <name>10.10.0.0/16</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* ssh servers */</junos:comment>
    <route>
      <name>10.17.136.0/24</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* Workstations */</junos:comment>
    <route>
      <name>10.150.0.0/16</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* Hosts */</junos:comment>
    <route>
      <name>10.157.64.0/19</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
    <junos:comment>/* Build Servers */</junos:comment>
    <route>
      <name>10.10.0.0/16</name>
      <next-hop><xsl:value-of select="$bkup-rtr"/></next-hop>
      <retain/>
      <no-readvertise/>
    </route>
  </static>
```

```
</routing-options>
<policy-options>
  <policy-statement>
    <name>redist</name>
    <from>
      <protocol>static</protocol>
    </from>
    <then>
      <accept/>
    </then>
  </policy-statement>
</policy-options>
<apply-groups>re0</apply-groups>
<apply-groups>re1</apply-groups>
<groups>
  <name>re0</name>
  <system>
    <host-name>
      <xsl:value-of select="$hostname"/></host-name>
  </system>
  <interfaces>
    <interface>
      <name>fxp0</name>
      <unit>
        <name>0</name>
        <family>
          <inet>
            <address>
              <name>
                <xsl:value-of select="$fxp0-addr"/>
              </name>
            </address>
          </inet>
        </family>
      </unit>
    </interface>
  </interfaces>
</groups>
<groups>
  <name>re1</name>
</groups>
<interfaces>
  <interface>
    <name>lo0</name>
    <unit>
      <name>0</name>
      <family>
        <inet>
          <address>
            <name><xsl:value-of select="$rid"/></name>
          </address>
        </inet>
      </family>
    </unit>
  </interface>
</interfaces>
```

```
            </transient-change>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

var $macro-name = 'config-system.xsl';
match configuration {
  var $rid = routing-options/router-id;
  for-each (apply-macro[name = 'config-system']) {
    var $hostname = data[name = 'host-name']/value;
    var $fxp0-addr = data[name = 'mgmt-address']/value;
    var $backup-router = data[name = 'backup-router']/value;
    var $bkup-rtr = {
      if ($backup-router) {
        expr $backup-router;
      }
      else {
        var $fxp01 = substring-before($fxp0-addr,                    '.');
        var $fxp02 = substring-before(substring-after($fxp0-addr, '.'), '.');
        var $fxp03 = substring-before(substring-after(substring-after(
                    $fxp0- addr, '.'), '.'), '.');
        var $plen = substring-after($fxp0-addr, '/');
        if ($plen = 22) {
          expr $fxp01 _ '.' _ $fxp02 _ '.' _ $fxp03 div 4 * 4 + 3 _ '.254';
        }
        else if ($plen = 24) {
          expr $fxp01 _ '.' _ $fxp02 _ '.' _ $fxp03 _ '.254';
        }
      }
    }
    if (not($rid) or not($hostname) or not($fxp0-addr)) {
      <xnm:error> {
        <message> "Must set router ID, host-name, and mgmt-address to use
                  this script.";
      }
    }
    else {
      <transient-change> {
        <system> {
          /* Set the following */
          <domain-name> "your-domain.net";
          <domain-search> "domain.net";
          <backup-router> {
            <address> $bkup-rtr;
          }
          <time-zone> "America/Los_Angeles";
          <authentication-order> "radius";
          <authentication-order> "password";
```

```
  <root-authentication> {
    <encrypted-password>
            "$1$Q3CG88jZ$.qhPUZaHdaIMWF2CvxKTe0";
  }
  <name-server> {
    <name> "192.168.5.68";
  }
  <name-server> {
    <name> "172.17.28.100";
  }
  <radius-server> {
    <name> "192.168.170.241";
    <secret> "$9$4xoDk5T3n/AHkmTQFCA0BIclKWL7sgaRh-bs4GU";
  }
  <radius-server> {
    <name> "192.168.4.240";
    <secret> "$9$TQ/t1IcSrKAt0IRheK8X7VYgaZDm5zNdiqmTn6";
  }
  <login> {
    <class> {
      <permissions> "all";
    }
    <user> {
      <name> "johnny";
      <uid> "928";
      <class> "superuser";
      <authentication> {
        <encrypted-password>"$1$kPU..$w.4FGRAGanJ8U4Yq6sbj7.";
    }
  }
}
<services> {
  <finger>;
  <ftp>;
  <ssh>;
  <telnet>;
  <xnm-clear-text>;
}
<syslog> {
  <user> {
    <name> "*";
    <contents> {
      <name> "any";
      <emergency>;
    }
  }
  <host> {
    <name> "host1";
    <contents> {
      <name> "any";
      <notice>;
    }
    <contents> {
      <name> "interactive-commands";
      <any>;
    }
```

```
  }
  <file> {
    <name> "messages";
    <contents> {
      <name> "any";
      <notice>;
    }
    <contents> {
      <name> "any";
      <warning>;
    }
    <contents> {
      <name> "authorization";
      <info>;
    }
    <archive> {
      <world-readable>;
    }
  }
  <file> {
    <name> "security";
    <contents> {
      <name> "interactive-commands";
      <any>;
    }
    <archive> {
      <world-readable>;
    }
  }
}
<processes> {
  <routing> {
    <undocumented><enable>;
  }
  <snmp> {
    <undocumented><enable>;
  }
  <ntp> {
    <undocumented><enable>;
  }
  <inet-process> {
    <undocumented> <enable>;
  }
  <mib-process> {
    <undocumented> <enable>;
  }
  <undocumented><management> {
    <enable>;
  }
  <watchdog> {
    <enable>;
  }
  <ntp> {
    <boot-server> "domain.net";
    <server> {
      <name> "domainr.net";
```

```
    }
  }
}
<snmp> {
  <location> "Software lab";
  <contact> "Michael Landon";
  <interface> "fxp0.0";
  <community> {
    <name> "public";
    <authorization> "read-only";
    <clients> {
      <name> "0.0.0.0/0";
      <restrict>;
    }
    <clients> {
      <name> "192.168.1.252/32";
    }
    <clients> {
      <name> "10.197.169.222/32";
    }
    <clients> {
      <name> "10.197.169.188/32";
    }
    <clients> {
      <name> "10.197.169.193/32";
    }
    <clients> {
      <name> "192.168.65.46/32";
    }
    <clients> {
      <name> "10.209.152.0/23";
    }
  }
  <community> {
    <name> "private";
    <authorization> "read-write";
    <clients> {
      <name> "0.0.0.0/0";
      <restrict>;
    }
    <clients> {
      <name> "10.197.169.188/32";
    }
  }
}
<routing-options> {
  <static> {
    <junos:comment> "/* safety precaution */";
    <route> {
      <name> "0.0.0.0/0";
      <discard>;
      <retain>;
      <no-readvertise>;
    }
    <junos:comment> "/* corporate net */";
    <route> {
```

```
        <name> "172.16.0.0/12";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* lab nets */";
      <route> {
        <name> "192.168.0.0/16";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* reflector */";
      <route> {
        <name> "10.17.136.192/32";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* another lab1*/";
      <route> {
        <name> "10.10.0.0/16";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* ssh servers */";
      <route> {
        <name> "10.17.136.0/24";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* Workstations */";
      <route> {
        <name> "10.150.0.0/16";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* Hosts */";
      <route> {
        <name> "10.157.64.0/19";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
      <junos:comment> "/* Build Servers */";
      <route> {
        <name> "10.10.0.0/16";
        <next-hop> $bkup-rtr;
        <retain>;
        <no-readvertise>;
      }
    }
  }
```

```
            <policy-options> {
              <policy-statement> {
                <name> "redist";
                <from> {
                  <protocol> "static";
                }
                <then> {
                  <accept>;
                }
              }
            }
            <apply-groups> "re0";
            <apply-groups> "re1";
            <groups> {
              <name> "re0";
              <system> {
                <host-name> $hostname;
              }
              <interfaces> {
                <interface> {
                  <name> "fxp0";
                  <unit> {
                    <name> "0";
                    <family> {
                      <inet> {
                        <address> {
                          <name> $fxp0-addr;
                        }
                      }
                    }
                  }
                }
              }
            }
            <groups> {
              <name> "re1";
            }
            <interfaces> {
              <interface> {
                <name> "lo0";
                <unit> {
                  <name> "0";
                  <family> {
                    <inet> {
                      <address> {
                        <name> $rid;
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **config-system.xsl** or **config-system.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **config-system.slax**.

   ```
   system {
     scripts {
       commit {
         allow-transients;
         file config-system.xsl;
       }
     }
   }
   apply-macro config-system {
     host-name test;
     mgmt-address 10.0.0.1/32;
     backup-router 10.0.0.2;
   }
   ```

   The **host-name** and **mgmt-address** statements are mandatory. The **backup-router** statement is optional. You can substitute a hostname, a management Ethernet (**fxp0**) IP address, and a backup router IP address that are appropriate for your device.

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** After committing the configuration, issue the **show | display commit-scripts** configuration mode command to view the device base configuration.

```
user@host# show | display commit-scripts
...
```

# Example: Prepending a Global Policy

This commit script example ensures that a BGP global import policy is applied to all your BGP imports before any other import policies are applied.



## Requirements

This example uses a device running Junos OS.

## Overview and Commit Script

For most configuration objects, the order in which the object or its children is created is not significant, because the Junos OS configuration management software stores and displays configuration objects in predetermined positions in the configuration hierarchy. However, some configuration objects—such as routing policies and firewall filters—consist of elements that must be processed and analyzed sequentially in order to produce the intended routing behavior.

This example commit script ensures that a BGP global import policy is applied to all your BGP imports before any other import policies are applied.

This example automatically prepends the **bgp_global_import** policy in front of any other BGP import policies. If the **bgp_global_import** policy statement is not included in the configuration, an error message is generated, and the commit operation fails.

Otherwise, the commit script uses the **insert="before"** Junos XML protocol attribute and the **position()** XSLT function to control the position of the global BGP policy in relation to any other applied policies. The **insert="before"** attribute inserts the **bgp_global_import** policy in front of the first preexisting BGP import policy.

If there is no preexisting default BGP import policy, the global policy is included in the configuration.

The example script is shown in both XSLT and SLAX syntax:

XSLT Syntax

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:if  test="not(policy-options/policy-statement[name='bgp_global_import'])">
      <xnm:error>
        <message>Policy error: Policy bgp_global_import required</message>
      </xnm:error>
    </xsl:if>
    <xsl:for-each select="protocols/bgp | protocols/bgp/group |
                          protocols/bgp/group/neighbor">
      <xsl:variable name="first" select="import[position() = 1]"/>
      <xsl:if test="$first">
        <xsl:call-template name="jcs:emit-change">
          <xsl:with-param name="tag" select="'transient-change'"/>
          <xsl:with-param name="content">
            <import insert="before"
                    name="{$first}">bgp_global_import</import>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:if>
    </xsl:for-each>
    <xsl:for-each select="protocols/bgp">
      <xsl:if test="not(import)">
        <xsl:call-template name="jcs:emit-change">
          <xsl:with-param name="tag" select="'transient-change'"/>
          <xsl:with-param name="content">
            <import>bgp_global_import</import>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

SLAX Syntax

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  if (not(policy-options/policy-statement[name='bgp_global_import'])) {
    <xnm:error> {
      <message> "Policy error: Policy bgp_global_import required";
    }
  }
  for-each (protocols/bgp | protocols/bgp/group |
            protocols/bgp/group/neighbor) {
    var $first = import[position() = 1];
```

```
    if ($first) {
      call jcs:emit-change($tag = 'transient-change') {
        with $content = {
          <import insert="before" name="{$first}"> "bgp_global_import";
        }
      }
    }
  }
  for-each (protocols/bgp) {
    if (not(import)) {
      call jcs:emit-change($tag = 'transient-change') {
        with $content = {
          <import> "bgp_global_import";
        }
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **bgp-global-import.xsl** or **bgp-global-import.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **bgp-global-import.slax**.

```
system {
  scripts {
    commit {
      allow-transients;
      file bgp-global-import.xsl;
    }
  }
}
interfaces {
  fe-0/0/0 {
    unit 0 {
      family inet {
        address 192.168.16.2/24;
      }
      family inet6 {
        address 2002:18a5:e996:beef::2/64;
      }
    }
  }
}
routing-options {
  autonomous-system 65400;
}
```

```
protocols {
  bgp {
    group fish {
      neighbor 192.168.16.4 {
        import [ blue green ];
        peer-as 65401;
      }
      neighbor 192.168.16.6 {
        peer-as 65402;
      }
    }
  }
}
policy-options {
  policy-statement blue {
    from protocol bgp;
    then accept;
  }
  policy-statement green {
    then accept;
  }
  policy-statement bgp_global_import {
    then accept;
  }
}
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Configuration

**Purpose** Verify that the script behaves as expected.

**Action** When you issue the **show protocols** configuration mode command, the **bgp_global_import** import policy is not displayed, because it is added as a transient change.

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

```
bgp {
  group fish {
    neighbor 192.168.16.4 {
      import [ blue green ];
      peer-as 65401;
    }
    neighbor 192.168.16.6 {
      peer-as 65402;
    }
  }
}
```

The commit script adds the **import bgp_global_import** statement at the **[edit protocols bgp]** hierarchy level and prepends the **bgp_global_import** policy to the **192.168.16.4** neighbor policy chain. Issue the **show protocols | display commit-scripts** to view all configuration statements including transient changes.

```
[edit]
user@host# show protocols | display commit-scripts
bgp {
  import bgp_global_import;
  group fish {
    neighbor 192.168.16.4 {
      import [ bgp_global_import blue green ];
      peer-as 65401;
    }
    neighbor 192.168.16.6 {
      peer-as 65402;
    }
  }
}
```

After you add a policy to the **192.168.16.6** neighbor, which previously had no policies applied, the **bgp_global_import** policy is prepended. Issue the **show protocols | display commit-scripts** command to view all configuration statements including transient changes.

```
[edit]
user@host# set protocols bgp group fish neighbor 192.168.16.6 import green
```

```
[edit]
user@host# show protocols | display commit-scripts
bgp {
  import bgp_global_import;
  group fish {
    neighbor 192.168.16.4 {
      import [ bgp_global_import blue green ];
      peer-as 65401;
    }
    neighbor 192.168.16.6 {
      import [ bgp_global_import green ];
      peer-as 65402;
    }
  }
}
```

## Example: Preventing Import of the Full Routing Table

In the Junos OS routing policy, if you configure a policy with no match conditions and a terminating action of **then accept**, and then apply the policy to a routing protocol, the protocol imports the entire routing table. This example shows how to use a commit script to prevent this scenario.



### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

This example inspects the **import** statements configured at the **[edit protocols ospf]** and **[edit protocols isis]** hierarchy levels to determine if any of the named policies contain a **then accept** term with no match conditions. The script protects against importing the full routing table into these interior gateway protocols (IGPs).

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:param name="po"
        select="commit-script-input/configuration/policy-options"/>
  <xsl:template match="configuration">
    <xsl:apply-templates select="protocols/ospf/import"/>
    <xsl:apply-templates select="protocols/isis/import"/>
  </xsl:template>
  <xsl:template match="import">
    <xsl:param name="test" select="."/>
    <xsl:for-each select="$po/policy-statement[name=$test]">
      <xsl:choose>
        <xsl:when test="then/accept and not(to) and not(from)">
          <xnm:error>
            <xsl:call-template name="jcs:edit-path">
              <xsl:with-param name="dot" select="$test"/>
            </xsl:call-template>
            <xsl:call-template name="jcs:statement">
              <xsl:with-param name="dot" select="$test"/>
            </xsl:call-template>
            <message>policy contains bare 'then accept'</message>
          </xnm:error>
```

```
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

param $po = commit-script-input/configuration/policy-options;
match configuration {
  apply-templates protocols/ospf/import;
  apply-templates protocols/isis/import;
}
match import {
  param $test = .;
  for-each ($po/policy-statement[name=$test]) {
    if (then/accept and not(to) and not(from)) {
      <xnm:error> {
        call jcs:edit-path($dot = $test);
        call jcs:statement($dot = $test);
        <message> "policy contains bare 'then accept'";
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **import.xsl** or **import.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.
2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **import.slax**.

```
system {
  scripts {
    commit {
      file import.xsl;
    }
  }
}
protocols {
  ospf {
    import bad-news;
  }
}
policy-options {
```

```
    policy-statement bad-news {
      then accept;
    }
  }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying the Commit Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration configures an **import** statement at the **[edit protocols ospf]** hierarchy level. Because the policy contains a **then accept** term with no match conditions, the script generates an error, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit protocols ospf]
  'import bad-news;'
  policy contains bare 'then accept'
error: 1 error reported by commit scripts
error: commit script failure
```

## Example: Requiring Internal Clocking on T1 Interfaces

This example shows how to use a commit script to require that T1 interfaces be configured with internal clocking.

## Requirements

This example uses a device running Junos OS with T1 interfaces.

## Overview and Commit Script

This commit script ensures that T1 interfaces are explicitly configured to use internal clocking. If the **clocking** statement is not included in the configuration, or if the **clocking external** statement is included, an error message is generated, and the configuration is not committed.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:junos="http://xml.juniper.net/junos/*/junos"
  xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
  xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:template match="configuration">
    <xsl:for-each select="interfaces/interface[starts-with(name, 't1-')]">
      <xsl:variable name="clock-source">
        <xsl:value-of select="clocking"/>
      </xsl:variable>
      <xsl:if test="not($clock-source = 'internal')">
      <!-- or xsl:if test="$clock-source != 'internal'" -->
        <xnm:error>
          <xsl:call-template name="jcs:edit-path"/>
          <xsl:call-template name="jcs:statement">
            <xsl:with-param name="dot" select="clocking"/>
          </xsl:call-template>
          <message>
            This T1 interface should have internal clocking.
          </message>
        </xnm:error>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  for-each (interfaces/interface[starts-with(name, 't1-')]) {
    var $clock-source = {
      expr clocking;
    }
    if (not($clock-source = 'internal')) {
      <xnm:error> {
```

```
            call jcs:edit-path();
            call jcs:statement($dot = clocking);
            <message> "This T1 interface should have internal clocking.";
          }
        }
      }
    }
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **clocking-error.xsl** or **clocking-error.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **clocking-error.slax**.

   ```
   system {
     scripts {
       commit {
         file clocking-error.xsl;
       }
     }
   }
   interfaces {
     t1-0/0/0 {
       clocking external;
     }
     t1-0/0/1 {
       unit 0;
     }
   }
   ```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying Commit Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The sample configuration stanzas configure two T1 interfaces **t1-0/0/0** and **t1-0/0/1**. Interface **t1-0/0/0** is configured with the **clocking external** statement, and interface **t1-0/0/1** does not include any **clocking** statement. The script generates an error, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit interfaces interface t1-0/0/0]
  'clocking external;'
  This T1 interface should have internal clocking.
[edit interfaces interface t1-0/0/1]
  ';'
  This T1 interface should have internal clocking.
error: 2 errors reported by commit scripts
error: commit script failure
```

# Example: Requiring and Restricting Configuration Statements

Junos OS commit scripts enforce custom configuration rules. When a candidate configuration is committed, it is inspected by each active commit script. This example uses a commit script to specify required and prohibited configuration statements.



## Requirements

This example uses a device running Junos OS that has the Ethernet management interface **fxp0**.

## Overview and Commit Script

This example shows how to use a commit script to specify required and prohibited configuration statements. The following commit script ensures that the Ethernet management interface (**fxp0**) is configured and detects when the interface is improperly disabled. The script also detects when the **bgp** statement is not included at the **[edit protocols]** hierarchy level. In all cases, the script generates an error message, and the commit operation fails.

The example script is shown in both XSLT and SLAX syntax:

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:junos="http://xml.juniper.net/junos/*/junos"
xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
<xsl:import href="../import/junos.xsl"/>

<xsl:template match="configuration">
  <xsl:call-template name="error-if-missing">
    <xsl:with-param name="must"
          select="interfaces/interface[name='fxp0']/
          unit[name='0']/family/inet/address"/>
    <xsl:with-param name="statement"
          select="'interfaces fxp0 unit 0 family inet address'"/>
  </xsl:call-template>
  <xsl:call-template name="error-if-present">
    <xsl:with-param name="must"
          select="interfaces/interface[name='fxp0']/disable
          | interfaces/interface[name='fxp0']/
          unit[name='0']/disable"/>
    <xsl:with-param name="message">
      <xsl:text>The fxp0 interface is disabled.</xsl:text>
    </xsl:with-param>
  </xsl:call-template>
  <xsl:call-template name="error-if-missing">
    <xsl:with-param name="must" select="protocols/bgp"/>
    <xsl:with-param name="statement" select="'protocols bgp'"/>
  </xsl:call-template>
</xsl:template>
<xsl:template name="error-if-missing">
  <xsl:param name="must"/>
  <xsl:param name="statement" select="'unknown'"/>
  <xsl:param name="message"
          select="'missing mandatory configuration statement'"/>
  <xsl:if test="not($must)">
    <xnm:error>
      <edit-path><xsl:copy-of select="$statement"/></edit-path>
      <message><xsl:copy-of select="$message"/></message>
    </xnm:error>
  </xsl:if>
</xsl:template>
<xsl:template name="error-if-present">
  <xsl:param name="must" select="1"/> <!-- give error if param missing -->
  <xsl:param name="message" select="'invalid configuration statement'"/>
  <xsl:for-each select="$must">
    <xnm:error>
      <xsl:call-template name="jcs:edit-path"/>
      <xsl:call-template name="jcs:statement"/>
      <message><xsl:copy-of select="$message"/></message>
    </xnm:error>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
```

```
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match configuration {
  call error-if-missing($must =
        interfaces/interface[name='fxp0']/unit[name='0']/family/inet/address,
        $statement = 'interfaces fxp0 unit 0 family inet address');
  call error-if-present($must = interfaces/interface[name='fxp0']/disable |
        interfaces/interface[name='fxp0']/unit[name='0']/disable) {
    with $message = {
      expr "The fxp0 interface is disabled.";
    }
  }
  call error-if-missing($must = protocols/bgp, $statement = 'protocols bgp');
}
error-if-missing ($must, $statement = 'unknown', $message =
        'missing mandatory configuration statement') {
  if (not($must)) {
    <xnm:error> {
      <edit-path> {
        copy-of $statement;
      }
      <message> {
        copy-of $message;
      }
    }
  }
}
error-if-present ($must = 1, $message = 'invalid configuration statement') {
  for-each ($must) {
    <xnm:error> {
      call jcs:edit-path();
      call jcs:statement();
      <message> {
        copy-of $message;
      }
    }
  }
}
```

## Configuration

**Step-by-Step Procedure**

To download, enable, and test the script:

1. Copy the XSLT or SLAX script into a text file, name the file **no-nukes.xsl** or **no-nukes.slax** as appropriate, and copy it to the **/var/db/scripts/commit/** directory on the device.

2. Select the following test configuration stanzas, and press Ctrl+c to copy them to the clipboard.

   If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **no-nukes.slax**.

```
system {
  scripts {
```

```
        commit {
          file no-nukes.xsl;
        }
      }
    }
    interfaces {
      fxp0 {
        disable;
        unit 0 {
          family inet {
            address 10.0.0.1/24;
          }
        }
      }
    }
```

3. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your device configuration.

   ```
   [edit]
   user@host# load merge terminal
   [Type ^D at a new line to end input]
   ... Paste the contents of the clipboard here ...
   ```

   a. At the prompt, paste the contents of the clipboard by using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl+d.

4. Issue the **commit** command to commit the configuration.

   ```
   user@host# commit
   ```

## Verification

### Verifying Commit Script Execution

**Purpose** Verify that the script behaves as expected.

**Action** Review the output of the **commit** command. The script requires that the Ethernet management interface (**fxp0**) is configured and enabled and that the **bgp** statement is included at the **[edit protocols]** hierarchy level. The sample configuration stanzas include the **fxp0** interface but disable it. In addition, the **bgp** statement is not configured at the **[edit protocols]** hierarchy level. When you run the script, it generates an error, and the commit operation fails. The following output appears after issuing a **commit** command:

```
[edit]
user@host# commit
[edit interfaces interface fxp0 disable]
  'disable;'
  The fxp0 interface is disabled.
protocols bgp
  missing mandatory configuration statement
error: 2 errors reported by commit scripts
```

```
error: commit script failure
```

CHAPTER 10

# Summary of Configuration Automation Configuration Statements

## allow-transients

**Syntax** allow-transients;

**Hierarchy Level** [edit system scripts commit]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, enable transient configuration changes to be committed.

**Default** Transient changes are disabled by default. If you do not include the **allow-transients** statement, and an enabled script generates transient changes, the command-line interface (CLI) generates an error message and the commit operation fails.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Generating a Persistent or Transient Change on page 59
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 75

## apply-macro

**Syntax**

```
apply-macro apply-macro-name {
  parameter-name parameter-value;
}
```

**Hierarchy Level** All hierarchy levels

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** With commit script macros, use custom syntax in your configuration.

Macros work by locating **apply-macro** statements that you include in the candidate configuration and using the values specified in the **apply-macro** statement as parameters to a set of instructions (the macro) defined in a commit script. The commit script alters your configuration from one that contains custom syntax into a full configuration containing standard Junos OS statements.

In effect, your custom configuration syntax serves a dual purpose. The syntax allows you to simplify your configuration tasks, and it provides data (or *hooks*) that are used by a commit script macros.

You can include the **apply-macro** statement at any level of the configuration hierarchy. You can include multiple **apply-macro** statements at each level of the configuration hierarchy; however, each must have a unique name.

**Options** ***apply-macro-name***—Name of the **apply-macro** statement.

***parameter-name***—One or more parameters. Parameters can be any text you want to include in your configuration.

***parameter-value***—A value that corresponds to the parameter name. Parameter values can be any text you want to include in your configuration.

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

**Related Documentation**

- Overview of Creating Custom Configuration Syntax with Macros on page 19

## checksum

**Syntax** checksum (md5 | sha-256 | sha1) *hash*;

**Hierarchy Level** [edit event-options event-script file *filename*],
[edit system scripts commit file *filename*],
[edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 9.5.

**Description** For Junos OS commit scripts and op scripts, specify the MD5, SHA-1, or SHA-256 checksum hash. When it executes a local event, commit, or op script, Junos OS verifies the authenticity of the script by using the configured checksum hash.

**Options**

**md5 *hash***—MD5 checksum of this script.

**sha-256 *hash***—SHA-256 checksum of this script.

**sha1 *hash***—SHA-1 checksum of this script.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring Checksum Hashes for a Commit Script on page 41
- Configuring Checksum Hashes for an Event Script
- Configuring Checksum Hashes for an Op Script
- Executing an Op Script from a Remote Site
- file checksum md5 command in the *System Basics and Services Command Reference*
- file checksum sha-256 command in the *System Basics and Services Command Reference*
- file checksum sha1 command in the *System Basics and Services Command Reference*

## commit

**Syntax**

```
commit {
  allow-transients;
  direct-access;
  file filename {
    checksum (md5 | sha-256 | sha1) hash;
    optional;
    refresh;
    refresh-from url;
    source url;
  }
  refresh;
  refresh-from url;
  traceoptions {
    file <filename> <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
  }
}
```

**Hierarchy Level** [edit system scripts]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, configure the commit-time scripting mechanism.

**Options** The statements are explained separately.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Storing and Enabling Scripts

## direct-access

**Syntax** direct-access;

**Hierarchy Level** [edit system scripts commit]

**Release Information** Statement introduced in Junos OS Release 9.1.

**Description** Specify that commit scripts read input configurations directly from the database when inspecting these scripts for errors.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Executing Large Commit Scripts on page 42

## file (Commit Scripts)

**Syntax**

```
file filename {
  checksum (md5 | sha-256 | sha1) hash;
  optional;
  refresh;
  refresh-from url;
  source url;
}
```

**Hierarchy Level** [edit system scripts commit]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, enable a commit script that is located in the **/var/db/scripts/commit** directory.

**Options**

*filename*—Name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing a commit script.

The remaining statements are explained separately.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Controlling Execution of Commit Scripts During Commit Operations on page 38

## optional

**Syntax**

```
optional;
```

**Hierarchy Level** [edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, allow a commit operation to succeed even if the script specified in the **file** statement is missing from the **/var/db/scripts/commit** directory on the device.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Controlling Execution of Commit Scripts During Commit Operations on page 38

## refresh (Commit Scripts)

**Syntax** refresh;

**Hierarchy Level** [edit system scripts commit],
[edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, overwrite the local copy of all enabled commit scripts or a single enabled script located in the **/var/db/scripts/commit** directory with the copy located at the source URL, as specified in the **source** statement at the same hierarchy level.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using a Master Source Location for a Script
- refresh-from (Commit Scripts) on page 188
- source (Commit Scripts) on page 190

## refresh-from (Commit Scripts)

**Syntax** refresh-from *url*;

**Hierarchy Level** [edit system scripts commit],
[edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, overwrite the local copy of all enabled commit scripts or a single enabled script located in the **/var/db/scripts/commit** directory with the copy located at a URL other than the URL specified in the **source** statement.

**Options** *url*—The source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using an Alternate Source Location for a Script
- refresh (Commit Scripts) on page 188
- source (Commit Scripts) on page 190

## scripts

**Syntax**

```
scripts {
  commit {
    allow-transients;
    direct-access;
    file filename {
      checksum (md5 | sha-256 | sha1) hash;
      optional;
      refresh;
      refresh-from url;
      source url;
    }
    refresh;
    refresh-from url;
    traceoptions {
      file <filename> <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
      no-remote-trace;
    }
  }
  op {
    file filename {
      arguments {
        argument-name {
          description descriptive-text;
        }
      }
      checksum (md5 | sha-256 | sha1) hash;
      command filename-alias;
      description descriptive-text;
      refresh;
      refresh-from url;
      source url;
    }
    no-allow-url
    refresh;
    refresh-from url;
    traceoptions {
      file <filename> <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
      no-remote-trace;
    }
  }
}
```

**Hierarchy Level** [edit system]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit or op scripts, configure scripting mechanisms.

**Options** The statements are explained separately.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Storing and Enabling Scripts

## source (Commit Scripts)

**Syntax** source *url*;

**Hierarchy Level** [edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** For Junos OS commit scripts, specify the location of the source file for an enabled script located in the **/var/db/scripts/commit** directory. When you include the **refresh** statement at the same hierarchy level and commit the configuration, the local copy is overwritten by the version stored at the specified URL.

**Options** *url*—The source specified as an HTTP URL, FTP URL, or scp-style remote file specification.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Using a Master Source Location for a Script
- Overview of Updating Scripts from a Remote Source
- refresh (Commit Scripts) on page 188
- refresh-from (Commit Scripts) on page 188

## traceoptions (Commit and Op Scripts)

**Syntax**

```
traceoptions {
  file <filename> <files number> <size size> <world-readable | no-world-readable>;
  flag flag;
  no-remote-trace;
}
```

**Hierarchy Level**

[edit system scripts commit],
[edit system scripts op]

**Release Information**

Statement introduced in Junos OS Release 7.4.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description**

Define tracing operations for commit or op scripts.

**Default**

If you do not include this statement, no script-specific tracing operations are performed.

**Options**

***filename***—Name of the file to receive the output of the tracing operation. All files are placed in the directory **/var/log**. By default, commit script process tracing output is placed in the file **cscript.log** and op script process tracing is placed in the file **op-script.log**. If you include the **file** statement, you must specify a filename. To retain the default, you can specify **cscript.log** or **op-script.log** as the filename.

**files** ***number***—(Optional) Maximum number of trace files. When a trace file named ***trace-file*** reaches its maximum size, it is renamed and compressed to ***trace-file*.0.gz**. When ***trace-file*** again reaches its maximum size, ***trace-file*.0.gz** is renamed ***trace-file*.1.gz** and ***trace-file*** is renamed and compressed to ***trace-file*.0.gz**. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum number of files, you also must specify a maximum file size with the **size** option and a filename.

**Range:** 2 through 1000

**Default:** 10 files

***flag***—Tracing operation to perform. To specify more than one tracing operation, include multiple **flag** statements. You can include the following flags:

- **all**—Log all operations
- **events**—Log important events
- **input**—Log script input data
- **offline**—Generate data for offline development
- **output**—Log script output data
- **rpc**—Log script RPCs
- **xslt**—Log the XSLT library

**no-world-readable**—Restrict file access to owner. This is the default.

**size *size***—(Optional) Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named ***trace-file*** reaches this size, it is renamed and compressed to ***trace-file*.0.gz**. When ***trace-file*** again reaches its maximum size, ***trace-file*.0.gz** is renamed ***trace-file*.1.gz** and ***trace-file*** is renamed and compressed to ***trace-file*.0.gz**. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option and a filename.

**Syntax:** ***x*k** to specify KB, ***x*m** to specify MB, or ***x*g** to specify GB

**Range:** 10 KB through 1 GB

**Default:** 128 KB

**world-readable**—Enable unrestricted file access.

**Required Privilege Level**

maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Tracing Commit Script Processing on page 204
- Tracing Op Script Processing

CHAPTER 11

# Junos XML and XSLT Tag Elements Used in Commit Scripts

## <change> (XSLT)

**Usage**

```
<change>
  <!-- tag elements representing configuration statements to load -->
</change>
```

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** Request that the Junos XML protocol server load configuration data into the candidate configuration by enclosing the configuration data within an opening **<change>** tag and closing **</change>** tag. Inside the **<change>** element, include the configuration data as Junos XML tag elements.

**Usage Guidelines** See "Overview of Generating Persistent or Transient Configuration Changes" on page 13 and "Overview of Creating Custom Configuration Syntax with Macros" on page 19.

**Related Documentation**

- <transient-change> (XSLT) on page 194

## <syslog> (Junos XML)

**Usage**

```
<syslog="namespace-URL" xmlns:xnm="namespace-URL">
  <message>syslog-message </message>
</syslog>
```

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** Record events that occur on a device running Junos OS.

**Attributes** **xmlns**—Names the Extensible Markup Language (XML) namespace for the contents of the tag element. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**xmlns:xnm**—Names the XML namespace for child tag elements that have the **xnm:** prefix on their names. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**Contents** **<message>**—Specifies the content of the system log message in a natural-language text string.

**Usage Guidelines** See "Generating a Custom Warning, Error, or System Log Message" on page 43.

## <transient-change> (XSLT)

**Usage**

```
<transient-change>
  <!-- tag elements representing configuration statements to load -->
</transient-change>
```

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** Request that the Junos XML protocol server load configuration data into the checkout configuration by enclosing the configuration data within an opening **<transient-change>** and closing **</transient-change>** tag. Inside the **<transient-change>** element, include the configuration data as Junos XML tag elements.

**Usage Guidelines** See "Overview of Generating Persistent or Transient Configuration Changes" on page 13 and "Overview of Creating Custom Configuration Syntax with Macros" on page 19.

**Related Documentation**

- <change> (XSLT) on page 193

## xnm:error (Junos XML)

**Usage**

```
<xnm:error xmlns="namespace-URL" xmlns:xnm="namespace-URL">
  <parse/>
  <source-daemon>module-name</source-daemon>
  <filename>filename</filename>
  <line-number>line-number</line-number>
  <column>column-number</column>
  <token>input-token-id</token>
  <edit-path>edit-path-name</edit-path>
  <statement>statement-string</statement>
  <message>error-string</message>
  <re-name>re-name-string</re-name>
  <database-status-information>user</database-status-information>
  <reason>reason-string</reason>
</xnm:error>
```

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** Indicate that the commit script has detected an error in the configuration and has caused the commit operation to fail. The child tag elements described in the Contents section detail the nature of the error.

**Attributes** **xmlns**—Names the XML namespace for the contents of the tag element. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**xmlns:xnm**—Names the XML namespace for child tag elements that have the **xnm:** prefix on their names. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**Contents**

**<column>**—Identifies the element that caused the error by specifying its position as the number of characters after the first character in the line specified by the **<line-number>** tag element in the configuration file that was being loaded (which is named in the **<filename>** tag element).

**<database-status-information>**—Provides information about the users currently editing the configuration.

**<edit-path>**—Specifies the command-line interface (CLI) configuration mode edit path in effect when the error occurred (provided only during loading of a configuration file).

**<filename>**—Names the configuration file that was being loaded.

**<line-number>**—Specifies the line number where the error occurred in the configuration file that was being loaded, which is named by the **<filename>** tag element.

**<message>**—Describes the error in a natural-language text string.

**<parse/>**—Indicates that there was a syntactic error in the request submitted by the client application.

**<re-name>**—Names the Routing Engine on which the <source-daemon> is running.

**<reason>**—Describes the reason for the error.

**<source-daemon>**—Names the Junos OS module that was processing the request in which the error occurred.

**<statement>**—Specifies the configuration statement in effect when the problem occurred.

**<token>**—Names the element in the request that caused the error.

**Usage Guidelines**

See "Generating a Custom Warning, Error, or System Log Message" on page 43.

**Related Documentation**

- xnm:warning (Junos XML) on page 195

## xnm:warning (Junos XML)

**Usage**

```
<xnm:warning xmlns="namespace-URL" xmlns:xnm="namespace-URL">
  <source-daemon>module-name</source-daemon>
  <filename>filename</filename>
  <line-number>line-number</line-number>
  <column>column-number</column>
  <token>input-token-id</token>
  <edit-path>edit-path-name</edit-path>
  <statement>statement-name</statement>
  <message>error-string</message>
  <reason>reason-string</reason>
```

```
</xnm:warning>
```

**Release Information** Statement introduced in Junos OS Release 7.4.

**Description** Indicate that the commit script has encountered a problem with the configuration. The child tag elements described in the Contents section detail the nature of the warning.

**Attributes** **xmlns**—Names the XML namespace for the contents of the tag element. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**xmlns:xnm**—Names the XML namespace for child tag elements that have the **xnm:** prefix on their names. The value is a URL of the form **http://xml.juniper.net/xnm/*version*/xnm**, where ***version*** is a string such as **1.1**.

**Contents** **<column>**—Identifies the element that caused the warning by specifying its position as the number of characters after the first character in the line specified by the **<line-number>** tag element in the configuration file that was being loaded (which is named in the **<filename>** tag element).

**<edit-path>**—Specifies the CLI configuration mode edit path in effect when the problem occurred (provided only during loading of a configuration file).

**<filename>**—Names the configuration file that was being loaded.

**<line-number>**—Specifies the line number where the problem occurred in the configuration file that was being loaded, which is named by the **<filename>** tag element.

**<message>**—Describes the warning in a natural-language text string.

**<reason>**—Describes the reason for the warning.

**<source-daemon>**—Names the Junos OS module that was processing the request in which the problem occurred.

**<statement>**—Names the configuration statement in effect when the problem occurred.

**<token>**—Names which element in the request caused the warning.

**Usage Guidelines** See "Generating a Custom Warning, Error, or System Log Message" on page 43

**Related Documentation**

- xnm:error (Junos XML) on page 194.

PART 3

# Administration

CHAPTER 12

# Configuration and Operations Configuration Statements



## Any Hierarchy Level

The following statement can be added at any level of the configuration:

```
apply-macro apply-macro-name {
  parameter-name parameter-value;
}
```

## [edit system scripts] Hierarchy Level

The following statements can be configured at the **[edit system]** hierarchy level. This is not a comprehensive list of statements available at the **[edit system]** hierarchy level. For more information about system configuration, see the *Junos OS System Basics Configuration Guide*.

```
[edit system]
scripts {
  commit {
    allow-transients;
    direct-access;
    file filename {
      checksum (md5 | sha-256 | sha1) hash;
      optional;
      refresh;
      refresh-from url;
      source url;
    }
    refresh;
    refresh-from url;
    traceoptions {
      file <filename> <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
      no-remote-trace;
    }
  }
```

```
    op {
      file filename {
        arguments {
          argument-name {
            description descriptive-text;
          }
        }
        checksum (md5 | sha-256 | sha1) hash;
        command filename-alias;
        description descriptive-text;
        refresh;
        refresh-from url;
        source url;
      }
      no-allow-url
      refresh;
      refresh-from url;
      traceoptions {
        file <filename> <files number> <size size> <world-readable | no-world-readable>;
        flag flag;
        no-remote-trace;
      }
    }
}
```

PART 4

# Troubleshooting

CHAPTER 13

# Troubleshooting Commit Scripts



## Displaying Commit Script Output

Table 5 on page 203 summarizes the Junos OS command-line interface (CLI) commands you can use to monitor and troubleshoot commit scripts. For more information about the **cscript.log** file, see "Tracing Commit Script Processing" on page 204.

### Table 5: Commit Script Configuration and Operational Mode Commands

| Task | Command |
|---|---|
| **Configuration Mode Commands** | |
| Display errors and warnings generated by commit scripts. | **commit** or **commit check** |
| Display detailed information. | **commit \| display detail** |
| Display the underlying Extensible Markup Language (XML) data. | **commit \| display xml** |
| Display the postinheritance contents of the configuration database. This view includes transient changes, but does not include changes made in configuration groups. | **show \| display commit-scripts** |
| Display the postinheritance contents of the configuration database. This view excludes transient changes. | **show \| display commit-scripts no-transients** |

Table 5: Commit Script Configuration and Operational Mode Commands *(continued)*

| Task | Command |
|---|---|
| Display the postinheritance configuration in XML format.<br><br>Viewing the configuration in XML format can be helpful when you are writing XML Path Language (XPath) expressions and configuration element tags. | **show \| display commit-scripts view** |
| Display the postinheritance configuration in XML format, but exclude transient changes. | **show \|**<br>**display commit-scripts view \|**<br>**display commit-scripts no-transients** |
| Display all configuration groups data, including script-generated changes to the groups. | **show groups \| display commit-scripts** |
| Display a particular configuration group, including script-generated changes to the group. | **show groups *group-name* \| display commit-scripts** |
| Operational Mode Commands | |
| Display logging data associated with all commit script processing. | **show log cscript.log** |
| Display processing for only the most recent commit operation. | **show log cscript.log \| last** |
| Display processing for script errors. | **show log cscript.log \| match error** |
| Display processing for a particular script. | **show log cscript.log \| match *filename*** |

**Related Documentation**

- Tracing Commit Script Processing on page 204

## Tracing Commit Script Processing

Commit script tracing operations track all commit script operations and record them in a log file. The logged error descriptions provide detailed information to help you solve problems faster.

The default operation of commit script tracing is to log important events in a file called **cscript.log** located in the **/var/log** directory on the device. When the file **cscript.log** reaches 128 kilobytes (KB), it is renamed with a number 0 through 9 (in ascending order) appended to the end of the file and then compressed. For example, the log file is saved as **cscript.log.0.gz**, then **cscript.log.1.gz** until there are 10 trace files. Then the oldest trace

file (**cscript.log.9.gz**) is overwritten. (For more information about how log files are created, see the *Junos OS System Log Messages Reference*.)

This section discusses the following topics:



## Minimum Configuration for Tracing for Commit Script Operations

If no commit script trace options are configured, the simplest way to view the trace output of a commit script is to configure the **output** trace flag and issue the **show log cscript.log | last** command. To do this, perform the following steps:

1. If you have not done so already, enable a commit script by including the **file** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   user@host# set file filename
   ```

2. Enable trace options by including the **traceoptions flag output** statement at the **[edit system scripts commit]** hierarchy level:

   ```
   [edit system scripts commit]
   user@host# set traceoptions flag output
   ```

3. Issue the **commit** command:

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

4. Display the resulting trace messages recorded in the file **/var/log/cscript.log**. At the end of the log is the output generated by the commit script you enabled in Step 1. To display the end of the log, issue the **show log cscript.log | last** operational mode command:

   ```
   [edit]
   user@host# run show log cscript.log | last
   ```

Table 6 on page 205 summarizes useful filtering commands that display selected portions of the **cscript.log** file.

### Table 6: Commit Script Tracing Operational Mode Commands

| Task | Command |
|---|---|
| Display logging data associated with all script processing. | **show log cscript.log** |
| Display script processing for only the most recent commit operation. | **show log cscript.log \| last** |
| Display processing for script errors. | **show log cscript.log \| match error** |
| Display script processing for a particular script. | **show log cscript.log \| match *filename*** |

### Example: Minimum Configuration for Enabling Traceoptions for Commit Scripts

Display the trace output for the commit script file **source-route.xsl**:

```
[edit]
system {
  scripts {
    commit {
      file source-route.xsl;
      traceoptions flag output;
    }
  }
}
```

```
[edit]
user@host# commit
[edit]
user@host# run show log cscript.log | last
Jun 20 10:21:24 summary: changes 0, transients 0 (allowed), syslog 0
Jun 20 10:24:15 commit script processing begins
Jun 20 10:24:15 reading commit script configuration
Jun 20 10:24:15 testing commit script configuration
Jun 20 10:24:15 opening commit script '/var/db/scripts/commit/source-route.xsl'
Jun 20 10:24:15 script file '/var/db/scripts/commit/source-route.xsl': size=699;
 md5 = d947972b429d17ce97fe987d94add6fd
Jun 20 10:24:15 reading commit script 'source-route.xsl'
Jun 20 10:24:15 running commit script 'source-route.xsl'
Jun 20 10:24:15 processing commit script 'source-route.xsl'
Jun 20 10:24:15 results of 'source-route.xsl'
Jun 20 10:24:15 begin dump
<commit-script-output xmlns:junos="http://xml.juniper.net/junos/*/junos"
xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
    <xnm:warning>
        <edit-path>[edit chassis]</edit-path>
        <message>IP source-route processing is not enabled.</message>
    </xnm:warning>
</commit-script-output>Jun 20 10:24:15 end dump
Jun 20 10:24:15 no errors from source-route.xsl
Jun 20 10:24:15 saving commit script changes
Jun 20 10:24:15 summary: changes 0, transients 0 (allowed), syslog 0
```

## Configuring Tracing of Commit Scripts

You cannot change the directory (**/var/log**) to which trace files are written. However, you can customize other trace file settings by including the following statements at the **[edit system scripts commit traceoptions]** hierarchy level:

```
[edit system scripts commit traceoptions]
file <filename> <files number> <size size> <world-readable | no-world-readable>;
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
```

```
no-remote-trace;
```

These statements are described in the following sections:



### Configuring the Commit Script Log Filename

By default, the name of the file that records trace output is **cscript.log**. You can specify a different name by including the **file** statement at the **[edit system scripts commit traceoptions]** hierarchy level:

```
[edit system scripts commit traceoptions]
file filename;
```

### Configuring the Number and Size of Commit Script Log Files

By default, when the trace file reaches 128 KB in size, it is renamed and compressed to ***filename*.0.gz**, then ***filename*.1.gz**, and so on, until there are 10 trace files. Then the oldest trace file (***filename*.9.gz**) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the **[edit system scripts commit traceoptions file <*filename*>]** hierarchy level:

```
[edit system scripts commit traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 640 KB and the maximum number of files to 20. When the file that receives the output of the tracing operation (***filename***) reaches 640 KB, it is renamed and compressed to ***filename*.0.gz**, and a new file called ***filename*** is created. When ***filename*** reaches 640 KB, ***filename*.0.gz** is renamed ***filename*.1.gz** and ***filename*** is renamed and compressed to ***filename*.0.gz**. This process repeats until there are 20 trace files. Then the oldest file (***filename*.19.gz**) is overwritten.

The number of files can range from 2 through 1000 files. The file size can range from 10 KB through 1 gigabyte (GB).

NOTE:

If you set either a maximum file size or a maximum number of trace files, you also must specify the other parameter and a filename.

### Configuring Access to Commit Script Log Files

By default, access to the commit script log file is restricted to the owner. You can manually configure access by including the **world-readable** or **no-world-readable** statement at the **[edit system scripts commit traceoptions file <*filename*>]** hierarchy level.

```
[edit system scripts commit traceoptions file <filename>]
```

(world-readable | no-world-readable);

The **no-world-readable** statement restricts commit script log access to the owner. The **world-readable** statement enables unrestricted access to the commit script log file.

### Configuring the Commit Script Trace Operations

By default, only important events are logged. You can configure the trace operations to be logged by including the following statements at the **[edit system scripts commit traceoptions]** hierarchy level:

```
[edit system scripts commit traceoptions]
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
```

Table 7 on page 208 describes the meaning of the commit script tracing flags.

**Table 7: Commit Script Tracing Flags**

| Flag | Description | Default Setting |
|---|---|---|
| **all** | Trace all operations. | Off |
| **events** | Trace important events. | On |
| **input** | Trace commit script input data. | Off |
| **offline** | Generate data for offline development. | Off |
| **output** | Trace commit script output data. | Off |
| **rpc** | Trace commit script RPCs. | Off |
| **xslt** | Trace the Extensible Stylesheet Language Transformations (XSLT) library. | Off |

## Troubleshooting Commit Scripts

After you enable a commit script and issue a **commit** command, the commit script takes effect immediately.

Table 8 on page 209 describes some common problems that might occur.

## Table 8: Troubleshooting Commit Scripts

| Problem | Solution |
|---|---|
| The output of the **commit check \| display detail** command does not reference the expected commit scripts. | Make sure you have enabled all the scripts by including the **file** statement for each one at the **[edit system scripts commit]** hierarchy level. |
| The output contains the error message:<br><br>error: could not open commit script:<br>/var/db/scripts/commit/*filename*:<br>No such file or directory | Make sure the file ***filename*** is in the **/var/db/scripts/commit/** directory on your switch, router, or security device. |
| The following error and warning messages appear:<br><br>error: invalid transient change generated by commit script: *filename*<br>warning: 1 transient change was generated without [system scripts commit allow-transients] | One of your commit scripts contains instructions to generate a transient change, but you have not enabled transient changes.<br><br>To rectify this problem, take one of the following actions:<br><br>• Remove the code that generates a transient change from the indicated script.<br>• Remove the script.<br>• Include the **allow-transients** statement at the **[edit system scripts commit]** hierarchy level. |
| An expected action does not occur.<br><br>For example, a warning message does not appear even though the configuration contains the problem that is supposed to evoke the warning message. | 1. Make sure you have enabled the script. Scripts are ignored if they are not enabled.<br>To enable a script, include the **file *filename*** statement at the **[edit system scripts commit]** hierarchy level.<br><br>2. Make sure you have included the required boilerplate in your script. For more information, see "Required Boilerplate for Commit Scripts" on page 30.<br><br>3. Make sure that the Extensible Markup Language Path (XPath) expressions in the script contain valid Junos OS command-line interface (CLI) statements expressed as Junos XML protocol tag elements.<br>You can verify the XML hierarchy by checking the *Junos XML API Configuration Reference* or by issuing the **show configuration \| display xml** operational mode command.<br><br>4. Make sure that the programming instructions in the script are referencing the correct context node.<br>If you nest one instruction inside another, the outer instruction changes the context node, so the inner instruction must be relative to the outer.<br>In the following example, the **<xsl:for-each>** instruction contains an XPath expression, which changes the context node. So the nested **<xsl:if>** instruction uses an XPath expression that is relative to the **interfaces/interface[starts-with(name, 't1-')]** XPath expression.<br><br><xsl:for-each<br>select="interfaces/interface[starts-with(name, 't1-')]"><br><xsl:if test="not(description)"> |

PART 5

# Index

# Index

## S



## T



## W



## X