JUNIPER NETWORKS

# Junos® OS

## Configuration and Operations Automation Guide

Release
13.1

Published: 2013-04-01

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



*Junos® OS Configuration and Operations Automation Guide*
13.1


The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

## 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.

# Abbreviated Table of Contents

# Table of Contents

# List of Figures

# List of Tables

# About This Guide

This preface provides the following guidelines for using the *Junos® OS Configuration and Operations Automation Guide*:



## Junos OS Documentation and Release Notes

For a list of related Junos OS documentation, see http://www.juniper.net/techpubs/software/junos/.

If the information in the latest release notes differs from the information in the documentation, follow the *Junos OS Release Notes.*

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at http://www.juniper.net/techpubs/.

Juniper Networks supports a technical book program to publish books by Juniper Networks engineers and subject matter experts with book publishers around the world. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration using the Junos operating system (Junos OS) and Juniper Networks devices. In addition, the Juniper Networks Technical Library, published in conjunction with O'Reilly Media, explores improving network security, reliability, and availability using Junos OS configuration techniques. All the books are for sale at technical bookstores and book outlets around the world. The current list can be viewed at http://www.juniper.net/books.

## Objectives

This guide provides an overview, instructions for using, and examples of Junos OS automation and the self-diagnosis features of Junos OS. Junos automation scripts, which include commit scripts, operation scripts, and event scripts, are based on Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX), the Junos Extensible Markup Language (XML) application programming interface (API), and the Junos XML Management Protocol. This guide also explains how to use commit script macros to provide simplified aliases for frequently used configuration statements and how to configure diagnostic event policies and actions associated with each policy.

NOTE: For additional information about the Junos OS—either corrections to or information that might have been omitted from this guide—see the software release notes at http://www.juniper.net/ .

## Audience

This guide is designed for network administrators who are configuring and monitoring a Juniper Networks M Series, MX Series, T Series, EX Series, or J Series router or switch.

To use this guide, you need a broad understanding of networks in general, the Internet in particular, networking principles, and network configuration. You must also be familiar with one or more of the following Internet routing protocols:

- Border Gateway Protocol (BGP)
- Distance Vector Multicast Routing Protocol (DVMRP)
- Intermediate System-to-Intermediate System (IS-IS)
- Internet Control Message Protocol (ICMP) router discovery
- Internet Group Management Protocol (IGMP)
- Multiprotocol Label Switching (MPLS)
- Open Shortest Path First (OSPF)
- Protocol-Independent Multicast (PIM)
- Resource Reservation Protocol (RSVP)
- Routing Information Protocol (RIP)
- Simple Network Management Protocol (SNMP)

Personnel operating the equipment must be trained and competent; must not conduct themselves in a careless, willfully negligent, or hostile manner; and must abide by the instructions provided by the documentation.

## Supported Platforms

For the features described in this manual, Junos OS currently supports the following platforms:

- ACX Series
- EX Series
- J Series
- M Series
- MX Series
- PTX Series
- QFX Series
- SRX Series
- T Series

## Using the Indexes

This reference contains two indexes: a standard index with topic entries, and an index of commands.

## Using the Examples in This Manual

If you want to use the examples in this manual, you can use the **load merge** or the **load merge relative** command. These commands cause the software to merge the incoming configuration into the current candidate configuration. The example does not become active until you commit the candidate configuration.

If the example configuration contains the top level of the hierarchy (or multiple hierarchies), the example is a *full example*. In this case, use the **load merge** command.

If the example configuration does not start at the top level of the hierarchy, the example is a *snippet*. In this case, use the **load merge relative** command. These procedures are described in the following sections.

### Merging a Full Example

To merge a full example, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration example into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following configuration to a file and name the file **ex-script.conf**. Copy the **ex-script.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   system {
     scripts {
   ```

```
        commit {
          file ex-script.xsl;
        }
      }
    }
    interfaces {
      fxp0 {
        disable;
        unit 0 {
          family inet {
            address 10.0.0.1/24;
          }
        }
      }
    }
```

2. Merge the contents of the file into your routing platform configuration by issuing the **load merge** configuration mode command:

```
[edit]
user@host# load merge /var/tmp/ex-script.conf
load complete
```

### Merging a Snippet

To merge a snippet, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration snippet into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following snippet to a file and name the file **ex-script-snippet.conf**. Copy the **ex-script-snippet.conf** file to the **/var/tmp** directory on your routing platform.

```
commit {
  file ex-script-snippet.xsl; }
```

2. Move to the hierarchy level that is relevant for this snippet by issuing the following configuration mode command:

```
[edit]
user@host# edit system scripts
[edit system scripts]
```

3. Merge the contents of the file into your routing platform configuration by issuing the **load merge relative** configuration mode command:

```
[edit system scripts]
user@host# load merge relative /var/tmp/ex-script-snippet.conf
load complete
```

For more information about the **load** command, see the CLI User Guide.

## Documentation Conventions

Table 1 on page xxvii defines notice icons used in this guide.

## Table 1: Notice Icons

| Icon | Meaning | Description |
|---|---|---|
|  | Informational note | Indicates important features or instructions. |
|  | Caution | Indicates a situation that might result in loss of data or hardware damage. |
|  | Warning | Alerts you to the risk of personal injury or death. |
|  | Laser warning | Alerts you to the risk of personal injury from a laser. |

Table 2 on page xxvii defines the text and syntax conventions used in this guide.

## Table 2: Text and Syntax Conventions

| Convention | Description | Examples |
|---|---|---|
| **Bold text like this** | Represents text that you type. | To enter configuration mode, type the**configure** command:<br><br>user@host> **configure** |
| `Fixed-width text like this` | Represents output that appears on the terminal screen. | `user@host>` **show chassis alarms**<br><br>`No alarms currently active` |
| *Italic text like this* | • Introduces or emphasizes important new terms.<br>• Identifies book names.<br>• Identifies RFC and Internet draft titles. | • A policy *term* is a named structure that defines match conditions and actions.<br>• *Junos OS System Basics Configuration Guide*<br>• RFC 1997, *BGP Communities Attribute* |
| *Italic text like this* | Represents variables (options for which you substitute a value) in commands or configuration statements. | Configure the machine's domain name:<br><br>[edit]<br>root@# **set system domain-name** *domain-name* |
| **Text like this** | Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components. | • To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level.<br>• The console port is labeled **CONSOLE**. |
| < > (angle brackets) | Enclose optional keywords or variables. | **stub <default-metric** ***metric*****>;** |

Table 2: Text and Syntax Conventions *(continued)*

| Convention | Description | Examples |
|---|---|---|
| \| (pipe symbol) | Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity. | **broadcast \| multicast**<br><br>**(*string1* \| *string2* \| *string3*)** |
| # (pound sign) | Indicates a comment specified on the same line as the configuration statement to which it applies. | **rsvp { # Required for dynamic MPLS only** |
| [ ] (square brackets) | Enclose a variable for which you can substitute one or more values. | **community name members [ *community-ids* ]** |
| Indention and braces ( { } ) | Identify a level in the configuration hierarchy. | [edit]<br>routing-options {<br>  static {<br>  route default {<br>  nexthop *address*;<br>  retain;<br>  }<br>  }<br>} |
| ; (semicolon) | Identifies a leaf statement at a configuration hierarchy level. | |
| J-Web GUI Conventions | | |
| **Bold text like this** | Represents J-Web graphical user interface (GUI) items you click or select. | • In the Logical Interfaces box, select **All Interfaces**.<br>• To cancel the configuration, click **Cancel**. |
| > (bold right angle bracket) | Separates levels in a hierarchy of J-Web selections. | In the configuration editor hierarchy, select **Protocols>Ospf**. |

## Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can send your comments to **techpubs-comments@juniper.net**, or fill out the documentation feedback form at https://www.juniper.net/cgi-bin/docbugreport/ . If you are using e-mail, be sure to include the following information with your comments:

- Document or topic name
- URL or page number
- Software release version (if applicable)

## Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or JNASC support contract,

or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at http://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf .
- Product warranties—For product warranty information, visit http://www.juniper.net/support/warranty/ .
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

## Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: http://www.juniper.net/customers/support/
- Search for known bugs: http://www2.juniper.net/kb/
- Find product documentation: http://www.juniper.net/techpubs/
- Find solutions and answer questions using our Knowledge Base: http://kb.juniper.net/
- Download the latest versions of software and review release notes: http://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: https://www.juniper.net/alerts/
- Join and participate in the Juniper Networks Community Forum: http://www.juniper.net/company/communities/
- Open a case online in the CSC Case Management tool: http://www.juniper.net/cm/

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://tools.juniper.net/SerialNumberEntitlementSearch/

## Opening a Case with JTAC

You can open a case with JTAC on the Web or by telephone.

- Use the Case Management tool in the CSC at http://www.juniper.net/cm/.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see http://www.juniper.net/support/requesting-support.html.

PART 1

# Overview

CHAPTER 1

# Overview of Configuration and Operations Automation

This chapter contains a brief overview of the configuration and operations automation tools provided by Juniper Networks Junos OS. These tools include commit scripts and macros, operation scripts, and event scripts and event policies.

## Junos Automation Overview

Junos automation consists of a suite of tools used to automate operational and configuration tasks on network devices running the Junos® operating system (Junos OS). The Junos automation tool kit is part of the standard Junos OS available on all switches, routers, and security devices running Junos OS. Junos automation tools, which leverage the native XML capabilities of Junos OS, include commit scripts, operation (op) scripts, event policies and event scripts, and macros.

Junos automation simplifies complex configurations and reduces potential configuration errors. It saves time by automating operational and configuration tasks. It also speeds troubleshooting and maximizes network uptime by warning of potential problems and automatically responding to system events.

Junos automation can capture the knowledge and expertise of experienced network operators and administrators and allow a business to leverage this combined expertise across the organization.

Junos automation scripts can be written in either of two scripting languages: Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX). XSLT is a standard for processing Extensible Markup Language (XML) data and is designed to convert one XML document into another. SLAX is an alternative to XSLT. It has a simple syntax that follows the style of C and PERL, but retains the same semantics as XSLT. Programmers who are familiar with C often find it easier to learn and use SLAX. Scripts written in one language are easily converted to the other.

The following sections describe the different types of functionality for Junos automation:


- Junos Configuration Automation: Commit Scripts on page 4
- Junos Operations Automation: Op Scripts on page 5
- Junos Event Automation: Event Scripts and Event Policy on page 5


### Junos Configuration Automation: Commit Scripts

Junos configuration automation uses commit scripts to automate the commit process. Junos OS commit scripts enforce custom configuration rules. 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 configuration rules
- Generate a commit error and halt the commit operation

Commit scripts, when used in conjunction with macros, allow you to simplify the Junos configuration and, at the same time, extend it with your own custom configuration syntax.

## Junos Operations Automation: Op Scripts

Junos operations automation uses op scripts to automate operational tasks and network troubleshooting. Junos OS op scripts can be executed manually in the CLI or upon user login, or they can be called from another script. Op scripts can process user arguments and can be constructed to:

- Create custom operational mode commands
- Execute a series of operational mode commands
- Customize the output of operational mode commands
- Shorten troubleshooting time by gathering operational information and iteratively narrowing down the cause of a network problem
- Perform controlled configuration changes
- Monitor the overall status of a device by creating a general operation script that periodically checks network warning parameters, such as high CPU usage.

## Junos Event Automation: Event Scripts and Event Policy

Junos event automation uses event policy and event scripts to instruct Junos OS to perform actions in response to system events.

### Event Policy

An event policy is an if-then-else construct that defines actions to be executed by the software on receipt of an event such as a system log message or SNMP trap. Event policies can be executed in response to a single system event or to correlated system events. For each policy, you can configure multiple actions including:

- Ignore the event
- Upload a file to a specified destination
- Execute Junos OS operational mode commands
- Execute Junos OS event scripts

### Event Scripts

Junos OS event scripts are triggered automatically by defined event policies in response to a system event and can instruct Junos OS to take immediate action. An event script automates network troubleshooting and network management by doing the following:

- Automatically diagnose and fix problems in the network
- Monitor the overall status of a device
- Run automatically as part of an event policy that detects periodic error conditions
- Change the configuration in response to a problem

**Related Documentation**

- Commit Script Overview on page 261

CHAPTER 2

# Scripts and Event Policy Configuration Statements

This chapter shows the complete configuration statement hierarchy for scripts and for event policy, listing all possible configuration statements and showing their level in the configuration hierarchy. When you are configuring Junos OS, your current hierarchy level is shown in the banner on the line preceding the **user@host#** prompt.

This chapter is organized as follows:



## 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 event-options] Hierarchy Level

The following statements can be included at the **[edit]** hierarchy level:

```
[edit]
event-options {
  destinations {
    destination-name {
      archive-sites {
        url <password password>;
      }
      transfer-delay seconds;
    }
  }
  event-script {
    file filename {
      checksum (md5 | sha-256 | sha1) hash;
      refresh;
      refresh-from url;
      remote-execution {
        remote-hostname {
          passphrase user-password;
          username user-login;
        }
      }
      source url;
    }
    max-datasize
    refresh;
    refresh-from url;
    traceoptions {
      file <filename> <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
      no-remote-trace;
    }
  }
  generate-event event-name {
    time-interval seconds;
    time-of-day hh:mm:ss;
  }
  max-policies
  policy policy-name {
    attributes-match {
      event1.attribute-name equals event2.attribute-name;
      event.attribute-name matches regular-expression;
      event1.attribute-name starts-with event2.attribute-name;
    }
    events [events];
    then {
      change-configuration {
        commands {
          "command";
        }
        commit-options {
```

```
            check <synchronize>;
            force;
            log "comment-string";
            synchronize;
          }
          retry count number interval seconds;
          user-name username;
        }
        event-script filename {
          arguments {
            argument-name argument-value;
          }
          destination destination-name {
            retry-count number retry-interval seconds;
            transfer-delay seconds;
          }
          output-filename filename;
          output-format (text | xml);
          user-name name;
        }
        execute-commands {
          commands {
            "command";
          }
          destination destination-name {
            retry-count count retry-interval seconds;
            transfer-delay seconds;
          }
          output-filename filename;
          output-format (text | xml);
          user-name username;
        }
        ignore;
        priority-override {
          facility facility-type;
          severity severity-level;
        }
        raise-trap;
        upload filename (filename | committed) destination destination-name {
          retry-count count retry-interval seconds;
          transfer-delay seconds;
          user-name username;
        }
      }
      within seconds {
        events [ events ];
        not events [ events ];
        trigger (after number | on number | until number);
      }
    }
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
    }
  }
```

## [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;
    }
    max-datasize
    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;
      max-datasize
      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;
    }
  }
}
```

CHAPTER 3

# Introduction to the Junos XML API and Junos XML Management Protocol

This chapter discusses the following topics:



## XML Overview

Extensible Markup Language (XML) is a language for defining a set of markers, called *tags*, that are applied to a data set or document to describe the function of individual elements and codify the hierarchical relationships between them. Tags look much like Hypertext Markup Language (HTML) tags, but XML is actually a metalanguage used to define tags that best suit the kind of data being marked.

For more details about XML, see *A Technical Introduction to XML* at http://www.xml.com/pub/a/98/10/guide0.html and the additional reference material at the http://www.xml.com site.

The official XML specification from the World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0*, is available at http://www.w3.org/TR/REC-xml .

The following sections discuss general aspects of XML:



## Tag Elements

XML has three types of tags: opening tags, closing tags, and empty tags. XML tag names are enclosed in angle brackets and are case sensitive. Items in an XML-compliant document or data set are always enclosed in paired opening and closing tags, and the tags must be properly nested. That is, you must close the tags in the same order in which you opened them. XML is stricter in this respect than HTML, which sometimes uses only opening tags. The following examples show paired opening and closing tags enclosing a value. The closing tags are indicated by the forward slash at the start of the tag name.

```
<interface-state>enabled</interface-state>
<input-bytes>25378</input-bytes>
```

The term *tag element* refers to a three-part set: opening tag, contents, and closing tag. The content can be an alphanumeric character string as in the preceding examples, or can itself be a *container* tag element, which contains other tag elements. For simplicity, the term *tag* is often used interchangeably with *tag element* or *element*.

If a tag element is *empty*—has no contents—it can be represented either as paired opening and closing tags with nothing between them, or as a single tag with a forward slash after the tag name. For example, the notation **<snmp-trap-flag/>** is equivalent to **<snmp-trap-flag></snmp-trap-flag>**.

As the preceding examples show, angle brackets enclose the name of the tag element. 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.

Junos XML and Junos XML protocol tag elements obey the XML convention that the tag element name indicates the kind of information enclosed by the tags. For example, the name of the Junos XML **<interface-state>** tag element indicates that it contains a description of the current status of an interface on the device, whereas the name of the**<input-bytes>** tag element indicates that its contents specify the number of bytes received.

When discussing tag elements in text, this documentation conventionally uses just the opening tag to represent the complete tag element (opening tag, contents, and closing tag). For example, the documentation refers to the **<input-bytes>** tag to indicate the entire **<input-bytes>*number-of-bytes*</input-bytes>** tag element.

## Attributes

XML elements can contain associated properties in the form of *attributes*, which specify additional information about an element. Attributes appear in the opening tag of an element and consist of an attribute name and value pair. The attribute syntax consists of the attribute name followed by an equals sign and then the attribute value enclosed in quotation marks. An XML element can have multiple attributes. Multiple attributes are separated by spaces and can appear in any order.

In the following example, the **configuration** element has two attributes, **junos:changed-seconds** and **junos:changed-localtime**.

```
<configuration junos:changed-seconds="1279908006"
junos:changed-localtime="2010-07-23 11:00:06 PDT">
```

The value of the **junos:changed-seconds** attribute is "1279908006", and the value of the **junos:changed-localtime** attribute is "2010-07-23 11:00:06 PDT".

## Namespaces

*Namespaces* allow an XML document to contain the same tag, attribute, or function names for different purposes and avoid name conflicts. For example, many namespaces may define a **print** function, and each may exhibit a different functionality. To use the functionality defined in one specific namespace, you must associate that function with the namespace that defines the desired functionality.

To refer to a tag, attribute, or function from a defined namespace, you must first provide the namespace Uniform Resource Identifier (URI) in your style sheet declaration . You then qualify a tag, attribute, or function from the namespace with the URI. Since a URI is often lengthy, generally a shorter prefix is mapped to the URI.

In the following example the **jcs** prefix is mapped to the namespace identified by the URI http://xml.juniper.net/junos/commit-scripts/1.0, which defines extension functions used in commit, op, and event scripts. The **jcs** prefix is then prepended to the **output** function, which is defined in that namespace.

```
<?xml version=”1.0”?>
 <xsl:stylesheet version=”1.0”
xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
   ...
   <xsl:value-of select="jcs:output('The VPN is up.')"/>
</xsl: stylesheet>
```

During processing, the prefix is expanded into the URI reference. Although there may be multiple namespaces that define an **output** element or function, the use of **jcs:output** explicitly defines which **output** function is used. You can choose any prefix to refer to the contents in a namespace, but there must be an existing declaration in the XML document that binds the prefix to the associated URI.

## Document Type Definition

An XML-tagged document or data set is *structured*, because a set of rules specifies the ordering and interrelationships of the items in it. The rules define the contexts in which each tagged item can—and in some cases must—occur. A file called a *document type definition*, or *DTD*, lists every tag element that can appear in the document or data set, defines the parent-child relationships between the tags, and specifies other tag characteristics. The same DTD can apply to many XML documents or data sets.

**Related Documentation**

- Junos XML API and Junos XML Management Protocol Overview on page 15
- XML and Junos OS on page 14

## XML and Junos OS

Extensible Markup Language (XML) is a standard for representing and communicating information. It is a metalanguage for defining customized tags that are applied to a data set or document to describe the function of individual elements and codify the hierarchical relationships between them. Junos OS natively supports XML for the operation and configuration of devices running Junos OS.

The Junos OS command-line interface (CLI) and the Junos OS infrastructure communicate using XML. When you issue an operational mode command in the CLI, the CLI converts the command into XML format for processing. After processing, Junos OS returns the output in the form of an XML document, which the CLI converts back into a readable format for display. Remote client applications also use XML-based data encoding for operational and configuration requests on devices running Junos OS.

The Junos XML API is an XML representation of Junos configuration statements and operational mode commands. It defines an XML equivalent for all statements in the Junos configuration hierarchy and many of the commands that you issue in CLI operational mode. Each operational mode command with a Junos XML counterpart maps to a request tag element and, if necessary, a response tag element.

To display operational mode command output as NETCONF and Junos XML tag elements instead of as the default formatted ASCII, issue the command, and pipe the output to the **display xml** command. Infrastructure tag elements in the response belong to the Junos XML management protocol instead of the NETCONF XML management protocol. The tag elements that describe Junos OS configuration or operational data belong to the Junos XML API, which defines the Junos content that can be retrieved and manipulated by both the Junos XML management protocol and the NETCONF XML management protocol operations. The following example compares the text and XML output for the **show chassis alarms** operational mode command:

```
user@host> show chassis alarms
No alarms currently active

user@host> show chassis alarms | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.4R1/junos">
    <alarm-information xmlns="http://xml.juniper.net/junos/10.4R1/junos-alarm">
        <alarm-summary>
            <no-active-alarms/>
        </alarm-summary>
    </alarm-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

To display the Junos XML API representation of any operational mode command, issue the command, and pipe the output to the **display xml rpc** command. The following example shows the Junos XML API tag element for the **show chassis alarms** command.

```
user@host> show chassis alarms | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.4R1/junos">
    <rpc>
```

```
        <get-alarm-information>
        </get-alarm-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

As shown in the previous example, the **| display xml rpc** option displays the command's corresponding Junos XML API request tag element that is sent to Junos OS for processing whenever the command is issued. In contrast, the **| display xml** option displays the actual output of the processed command in XML format.

When you issue the **show chassis alarms** operational mode command, the CLI converts the command into its equivalent Junos XML API request tag **<get-alarm-information>** and sends the XML request to the Junos infrastructure for processing. Junos OS processes the request and returns the **<alarm-information>** response tag element to the CLI. The CLI then converts the XML output into the "No alarms currently active" message that is displayed to the user.

Junos automation scripts use XML to communicate with the host device. Junos OS provides XML-formatted input to a script. The script processes the input source tree and then returns XML-formatted output to Junos OS. The script type determines the XML input document that is sent to the script as well as the output document that is returned to Junos OS for processing. Commit script input consists of an XML representation of the post-inheritance candidate configuration file. Event scripts receive an XML document containing the description of the triggering event. All script input documents contain a common node-set with information pertaining to the Junos OS environment.

**Related Documentation**

- *Junos XML API Configuration Reference*
- *Junos XML API Operational Reference*

## Junos XML API and Junos XML Management Protocol Overview

The Junos XML Management Protocol is an XML-based protocol that client applications use to request and change configuration information on routing, switching, and security platforms running Junos OS. It uses an XML-based data encoding for the configuration data and remote procedure calls. The protocol defines basic operations that are equivalent to configuration mode commands in the Junos OS command-line interface (CLI). Applications use the protocol operations to display, edit, and commit configuration statements (among other operations), just as administrators use CLI configuration mode commands such as **show**, **set**, and **commit** to perform those operations.

The Junos XML API is an XML representation of Junos configuration statements and operational mode commands. Junos XML configuration tag elements are the content to which the Junos XML protocol operations apply. Junos XML operational tag elements are equivalent in function to operational mode commands in the CLI, which administrators use to retrieve status information for a device.

Client applications request information and change the configuration on a device by encoding the request with tag elements from the Junos XML management protocol and

Junos XML API and sending it to the Junos XML protocol server on the device. The Junos XML protocol server is integrated into Junos OS and does not appear as a separate entry in process listings. The Junos XML protocol server directs the request to the appropriate software modules within the device, encodes the response in Junos XML and Junos XML protocol tag elements, and returns the result to the client application. For example, to request information about the status of a device's interfaces, a client application sends the Junos XML API **<get-interface-information>** request tag element . The Junos XML protocol server gathers the information from the interface process and returns it in the Junos XML API **<interface-information>** response tag element.

You can use the Junos XML management protocol and Junos XML API to configure devices running Junos OS or request information about the device configuration or operation. You can write client applications to interact with the Junos XML protocol server, and you can also utilize the Junos XML protocol to build custom end-user interfaces for configuration and information retrieval and display, such as a Web browser-based interface.

**Related Documentation**

- Advantages of Using the Junos XML Management Protocol and Junos XML API on page 16
- XML and Junos OS on page 14
- XML Overview on page 11

## Advantages of Using the Junos XML Management Protocol and Junos XML API

The Junos XML management protocol and Junos XML API fully document all options for every supported Junos operational request, all statements in the Junos configuration hierarchy, and basic operations that are equivalent to configuration mode commands. The tag names clearly indicate the function of an element in an operational or configuration request or a configuration statement.

The combination of meaningful tag names and the structural rules in a DTD makes it easy to understand the content and structure of an XML-tagged data set or document. Junos XML and Junos XML protocol tag elements make it straightforward for client applications that request information from a device to parse the output and find specific information.

### Parsing Device Output

The following example illustrates how the Junos XML API makes it easier to parse device output and extract the needed information. The example compares formatted ASCII and XML-tagged versions of output from a device running Junos OS.

The formatted ASCII follows:

```
Physical interface: fxp0, Enabled, Physical link is Up
  Interface index: 4, SNMP ifIndex: 3
```

The corresponding XML-tagged version is:

```
<interface>
  <name>fxp0</name>
```

```
    <admin-status>enabled</admin-status>
    <operational-status>up</operational-status>
    <index>4</index>
    <snmp-index>3</snmp-index>
  </interface>
```

When a client application needs to extract a specific value from formatted ASCII output, it must rely on the value's location, expressed either absolutely or with respect to labels or values in adjacent fields. Suppose that the client application wants to extract the interface index. It can use a regular-expression matching utility to locate specific strings, but one difficulty is that the number of digits in the interface index is not necessarily predictable. The client application cannot simply read a certain number of characters after the **Interface index:** label, but must instead extract everything between the label and the subsequent label **SNMP ifIndex:** and also account for the included comma.

A problem arises if the format or ordering of text output changes in a later version of the Junos OS. For example, if a **Logical index:** field is added following the interface index number, the new formatted ASCII might appear as follows:

```
Physical interface: fxp0, Enabled, Physical link is Up
  Interface index: 4, Logical index: 12, SNMP ifIndex: 3
```

An application that extracts the interface index number delimited by the **Interface index:** and **SNMP ifIndex:** labels now obtains an incorrect result. The application must be updated manually to search for the **Logical index:** label as the new delimiter.

In contrast, the structured nature of XML-tagged output enables a client application to retrieve the interface index by extracting everything within the opening **<index>** tag and closing **</index>** tag. The application does not have to rely on an element's position in the output string, so the Junos XML protocol server can emit the child tag elements in any order within the **<interface>** tag element. Adding a new **<logical-index>** tag element in a future release does not affect an application's ability to locate the **<index>** tag element and extract its contents.

## Displaying Device Output

XML-tagged output is also easier to transform into different display formats than formatted ASCII output. For instance, you might want to display different amounts of detail about a given device component at different times. When a device returns formatted ASCII output, you have to write special routines and data structures in your display program to extract and show the appropriate information for a given detail level. In contrast, the inherent structure of XML output is an ideal basis for a display program's own structures. It is also easy to use the same extraction routine for several levels of detail, simply ignoring the tag elements you do not need when creating a less detailed display.

**Related Documentation**

- Junos XML API and Junos XML Management Protocol Overview on page 15
- XML Overview on page 11

CHAPTER 4

# Understanding XSLT

This chapter contains some overview material, intended as a brief introduction to Extensible Stylesheet Language Transformations (XSLT) and XML Path Language (XPath). It is not a comprehensive user guide for XSLT or XPath. This chapter discusses the following topics:



## XSLT Overview

Commit scripts, op scripts, and event scripts can be written in Extensible Stylesheet Language Transformations (XSLT), which is a standard for processing Extensible Markup Language (XML) data. XSLT is developed by the World Wide Web Consortium (W3C) and is accessible at http://www.w3c.org/TR/xslt .



### XSLT Advantages

XSLT is a natural match for Junos OS, with its native XML capabilities. XSLT performs XML-to-XML transformations, turning one XML hierarchy into another. It offers a great degree of freedom and power in the way in which it transforms the input XML, allowing everything from making minor changes to the existing hierarchy (such as additions or deletions) to building a completely new document hierarchy.

Because XSLT was created to allow generic XML-to-XML transformations, it is a natural choice for both inspecting configuration syntax (which Junos OS can easily express in XML) and for generating errors and warnings (which Junos OS communicates internally as XML). XSLT includes powerful mechanisms for finding configuration statements that match specific criteria. XSLT can then generate the appropriate XML result tree from these configuration statements to instruct the Junos OS user-interface (UI) components to perform the desired behavior.

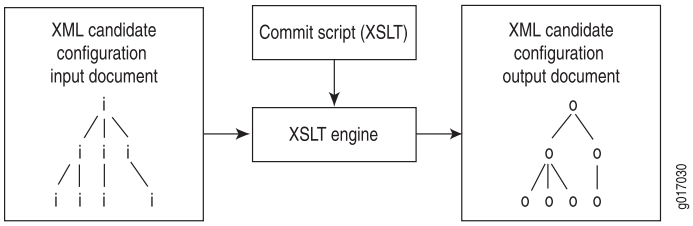
Although XSLT provides a powerful scripting ability, its focus is specific and limited. It does not make Junos OS vulnerable to arbitrary or malicious programmers. XSLT restricts programmers from performing haphazard operations, such as opening random Transmission Control Protocol (TCP) ports, forking numerous processes, or sending e-mail. The only action available in XSLT is to generate XML, and the XML is interpreted by the UI according to fixed semantics. An XSLT script can output only XML data, which is directly processed by the UI infrastructure to allow only the specific abilities listed above—generating error, warning, and system log messages, and persistent and transient configuration changes. This means that the impact of commit scripts, op scripts, and event scripts on the device is well-defined and can be viewed inside the command-line interface (CLI), using commands added for that purpose.

## XSLT Engine

XSLT is a language for transforming one XML document into another XML document. The basic model is that an XSLT engine (or processor) reads a script (or style sheet) and an XML document. The XSLT engine uses the instructions in the script to process the XML document by traversing the document's hierarchy. The script indicates what portion of the tree should be traversed, how it should be inspected, and what XML should be generated at each point. For commit scripts, op scripts, and event scripts, the XSLT engine is a function of the Junos OS management process (mgd).

Figure 1 on page 20 shows the relationship between an XSLT commit script and the XSLT engine.

### Figure 1: Flow of XSLT Commit Script Through the XSLT Engine




## XSLT Concepts

XSLT has seven basic concepts. These are summarized in Table 3 on page 20.

### Table 3: XSLT Concepts

| XSLT Concepts | Description |
|---|---|
| XPath | Expression syntax for specifying a node in the input document |

Table 3: XSLT Concepts *(continued)*

| XSLT Concepts | Description |
|---|---|
| Templates | Mechanism for mapping input hierarchies to instructions that handle them |
| Parameters | Mechanism for passing arguments to templates |
| Variables | Mechanism for defining read-only references to nodes |
| Programming instructions | Mechanism for defining logic in XSLT |
| Recursion | Mechanism by which templates call themselves to facilitate looping |
| Context (Dot) | Node currently being inspected in the input document |

**Related Documentation**

- XPath Overview on page 22
- XSLT Context (Dot) Overview on page 34
- XSLT Parameters Overview on page 26
- XSLT Programming Instructions Overview on page 30
- XSLT Recursion Overview on page 33
- XSLT Templates Overview on page 24
- XSLT Variables Overview on page 29

## XSLT Namespace

The XSLT namespace has the Uniform Resource Identifier (URI) http://www.w3.org/1999/XSL/Transform. The namespace must be included in the style sheet declaration of a script in order for the XSLT processor to recognize and use XSLT elements and attributes. The following example declares the XSLT namespace and associates the **xsl** prefix with the URI.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="route">
    ...
    </xsl:template>
</xsl:stylesheet
```

Once the XSLT namespace is declared in a script, you use elements and attributes from the namespace by adding the associated prefix, which in this case is **xsl**, to the tag or attribute name. In the preceding example, the XSLT processor knows to treat **xsl:template** as an XSLT instruction. During processing, the **xsl** prefix is expanded into the URI reference, and the functionality of the **template** element is defined by the XSLT namespace. For more information about namespaces, see "XML Overview" on page 11.

# XPath Overview

XSLT uses the XML Path Language (XPath) standard to specify and locate elements in the input document's XML hierarchy. XPath's powerful expression syntax enables you to define complex criteria for selecting portions of the XML input document.

## Nodes and Axes

XPath views every piece of the document hierarchy as a *node*. For commit scripts, op scripts, and event scripts, the important types of nodes are *element nodes*, *text nodes*, and *attribute nodes*. Consider the following XML tags:

```
<system>
  <host-name>my-router</host-name>
  <accounting inactive="inactive">
</system>
```

These XML tag elements show examples of the following types of XPath nodes:

- **<host-name>my-router</host-name>**—Element node
- **my-router**—Text node
- **inactive="inactive"**—Attribute node

Nodes are viewed as being arranged in certain *axes*. The *ancestor axis* points from a node up through its series of parent nodes. The *child axis* points through the list of an element node's direct child nodes. The *attribute axis* points through the list of an element node's set of attributes. The *following-sibling axis* points through the nodes that follow a node but are under the same parent. The *descendant axis* contains all the descendents of a node. There are numerous other axes that are not listed here.

Each XPath expression is evaluated from a particular node, which is referred to as the *context node* (or simply *context*). The context node is the node at which the XSLT processor is currently looking. XSLT changes the context as the document's hierarchy is traversed, and XPath expressions are evaluated from that particular context node.

NOTE: In Junos OS commit scripts, the context node concept corresponds to Junos OS hierarchy levels. For example, the **/configuration/system/domain-name** XPath expression sets the context node to the **[edit system domain-name]** hierarchy level.

We recommend including the **<xsl:template match="configuration">** template in all commit scripts. This element allows you to exclude the **/configuration/** root element from all XPath expressions in programming instructions (such as **<xsl:for-each>** or **<xsl:if>**) in the script, thus allowing you to begin XPath expressions at a Junos hierarchy level (for example, **system/domain-name**). For more information, see "Required Boilerplate for Commit Scripts" on page 270.

## Path and Predicate Syntax

An XPath expression contains two types of syntax, a path syntax and a predicate syntax. Path syntax specifies which nodes to inspect in terms of their path locations on one of the axes in the document's hierarchy from the current context node. Several examples of path syntax follow:

- **accounting-options**—Selects an element node named **accounting-options** that is a child of the current context.
- **server/name**—Selects an element node named **name** that is a child of an element named **server** that is a child of the current context.
- **/configuration/system/domain-name**—Selects an element node named **domain-name** that is the child of an element named **system** that is the child of the root element of the document (**configuration**).
- **parent::system/host-name**—Selects an element node named **host-name** that is the child of an element named **system** that is the parent of the current context node. The **parent::** axis can be abbreviated as two periods (**..**).

The predicate syntax allows you to perform tests at each node selected by the path syntax. Only nodes that pass the test are included in the result set. A predicate appears inside square brackets (**[ ]**) after a path node. Following are several examples of predicate syntax:

- **server[name = '10.1.1.1']**—Selects an element named **server** that is a child of the current context and has a child element named **name** whose value is 10.1.1.1.
- ***[@inactive]**—Selects any node (***** matches any node) that is a child of the current context and that has an attribute (**@** selects nodes from the **attribute** axis) named **inactive**.
- **route[starts-with(next-hop, '10.10.')]**—Selects an element named **route** that is a child of the current context and that has a child element named **next-hop** whose value starts with the string 10.10..

  The **starts-with** function is one of many functions that are built into XPath. XPath also supports relational tests, equality tests, and many more features not listed here.

## XPath Operators

XPath supports standard logical operators, such as **AND** and **|** (or); comparison operators, such as **=**, **!=**, **<**, and **>**; and numerical operators, such as **+**, **-**, and *****.

In XSLT, you always have to represent the less-than (**<**) operator as **<** and the less-than-or-equal-to (**<=**) operator as **<=** because XSLT scripts are XML documents, and less-than signs are represented this way in XML.

For more information about XPath functions and operators, consult a comprehensive XPath reference guide. XPath is fully described in the W3C specification at http://w3c.org/TR/xpath.

## XSLT Templates Overview

An XSLT script consists of one or more sets of rules called *templates*. Each template is a segment of code that contains rules to apply when a specified node is matched. You use the **<xsl:template>** element to build templates.

There are two types of templates, named and unnamed (or match), and they are described in the following sections.


- Unnamed (Match) Templates on page 24
- Named Templates on page 25


### Unnamed (Match) Templates

Unnamed templates, also known as match templates, include a **match** attribute that contains an XPath expression to specify the criteria for nodes upon which the template should be invoked. In the following example, the template applies to the element named **route** that is a child of the current context and that has a child element named **next-hop** whose value starts with the string **10.10.**.

```
<xsl:template match="route[starts-with(next-hop, '10.10.')]">
  <!-- ... body of the template ... -->
</xsl:template>
```

By default, when XSLT processes a document, it recursively traverses the entire document hierarchy, inspecting each node, looking for a template that matches the current node. When a matching template is found, the contents of that template are evaluated.

The **<xsl:apply-templates>** element can be used inside an unnamed template to limit and control XSLT's default, hierarchical traversal of nodes. If the **<xsl:apply-templates>** element has a **select** attribute, only nodes matching the XPath expression defined by the attribute are traversed. Otherwise all children of the context node are traversed. If the **select** attribute is included, but does not match any nodes, nothing is traversed and nothing happens.

In the following example, the template rule matches the **<route>** element in the XML hierarchy. All the nodes containing a **changed** attribute are processed. All **<route>** elements containing a **changed** attribute are replaced with a **<new>** element.

```
<xsl:template match="route">
  <new>
    <xsl:apply-templates select="*[@changed]"/>
  </new>
</xsl:template>
```

Using unnamed templates allows the script to ignore the location of a tag in the XML hierarchy. For example, if you want to convert all **<author>** tags into ** ** tags, using templates enables you to write a single rule that converts all **<author>** tags, regardless of their location in the input XML document.

For more information about how unnamed templates are used in scripts, see xsl:template match="/" Template.

## Named Templates

Named templates operate like functions in traditional programming languages, although with a verbose syntax. When the complexity of a script increases or a code segment appears in multiple places, you can modularize the code and create named templates. Like functions, named templates accept arguments and run only when explicitly called.

You create a named template by using the **<xsl:template>** element and defining the **name** attribute, which is similar to a function name in traditional programming languages. Use the **<xsl:param>** tag and its **name** attribute to define parameters for the named template, and optionally include the **select** attribute to declare default values for each parameter. The **select** attribute can contain XPath expressions. If the **select** attribute is not defined, the parameter defaults to an empty string.

The following example creates a template named **my-template** and defines three parameters, one of which defaults to the string **false**, and one of which defaults to the contents of the element node named **name** that is a child of the current context node. If the script calls the template and does not pass in a parameter, the default value is used.

```
<xsl:template name="my-template">
  <xsl:param name="a"/>
  <xsl:param name="b" select="'false'"/>
  <xsl:param name="c" select="name"/>
   <!-- ... body of the template ... -->
</xsl:template>
```

To invoke a named template in a script, use the **<xsl:call-template>** element. The **name** attribute is required and defines the name of the template being called. When processed, the **<xsl:call-template>** element is replaced by the contents of the **<xsl:template>** element it names.

When you invoke a named template, you can pass arguments into the template by including the **<xsl:with-param>** child element and specifying the **name** attribute. The value of the **<xsl:with-param> name** attribute must match a parameter defined in the actual template; otherwise the parameter is ignored. Optionally, you can set a value for each parameter with either the **select** attribute or the content of the **<xsl:with-param>** element. If you do not define a value for the parameter in the calling environment, the script passes in the current value of the parameter if it was previously initialized, or it generates an error if the parameter was never declared. For more information about passing parameters, see "XSLT Parameters Overview" on page 26.

In the following example, the template **my-template** is called with the parameter **c** containing the contents of the element node named **other-name** that is a child of the current context node.

```
<xsl:call-template name="my-template">
  <xsl:with-param name="c" select="other-name"/>
</xsl:call-template>
```

For an example showing how to use named templates in a commit script, see "Example: Requiring and Restricting Configuration Statements" on page 435.

**Related Documentation**

- XSLT Parameters Overview on page 26
- xsl:apply-templates on page 149
- xsl:call-template on page 149
- xsl:param on page 155
- xsl:template on page 157
- xsl:template match="/" Template on page 135
- xsl:with-param on page 161

## XSLT Parameters Overview

Parameters can be passed to either named or unnamed templates. Inside the template, parameters must be declared and can then be referenced by prefixing their name with the dollar sign ($).

### Declaring Parameters

The scope of a parameter can be global or local. A parameter whose value is set by Junos at script initialization must be defined as a global parameter. Global parameter declarations are placed just after the style sheet declarations. A script can assign a default value to the global parameter, which is used in the event that Junos does not give a value to the parameter.

```
<?xml version="1.0" standalone="yes"?>
  <xsl stylesheet 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"
    xmlns:ext="http://xmlsoft.org/XSLT/namespace" version="1.0">

<!-- global parameter -->
<xsl:param name="interface1"/>
```

Local parameters must be declared at the beginning of a block and their scope is limited to the block in which they are declared. Inside a template, you declare parameters using the **<xsl:param>** tag and **name** attribute. Optionally, declare default values for each parameter by including the **select** attribute, which can contain XPath expressions. If a template is invoked without the parameter, the default expression is evaluated, and the results are assigned to the parameter. If you do not define a default value in the template, the parameter defaults to an empty string.

The following named template **print-host-name** declares the parameter **message** and defines a default value:

```
<xsl:template name="print-host-name">
  <xsl:param name="message"
        select="concat('host-name: ', system/host-name)"/>
  <xsl:value-of select="$message"/>
</xsl:template>
```

The template accesses the value of the **message** parameter by prefixing the parameter name with the dollar sign ($).

## Passing Parameters

When you invoke a template, you pass arguments into the template using the **<xsl:with-param>** element and **name** attribute. The value of the **<xsl:with-param> name** attribute must match the name of a parameter defined in the actual template; otherwise the parameter is ignored. Optionally, for each parameter you pass to a template, you can define a value using either the **select** attribute or the contents of the **<xsl:with-param>** element.

The parameter value that gets used in a template depends on how the template is called. The following three examples, which call the **print-host-name** template, illustrate the possible calling environments.

If you call a template but do not include the **<xsl:with-param>** element for a specific parameter, the default expression defined in the template is evaluated, and the results are assigned to the parameter. If there is no default value for that parameter in the template, the parameter defaults to an empty string. The following example calls the named template **print-host-name** but does not include any parameters in the call. In this case, the named template will use the default value for the **message** parameter that was defined in the **print-host-name** template, or an empty string if no default exists.

```
<xsl:template match="configuration">
    <xsl:call-template name="print-host-name"/>
</xsl:template>
```

If you call a template and include a parameter, but do not define a value for the parameter in the calling environment, the script passes in the current value of the parameter if it was previously initialized, or it generates an error if the parameter was never declared. The following example calls the named template **print-host-name** and passes in the **message** parameter, but does not include a value. If **message** is declared and initialized in the script, and the scope is visible to the block, the current value of **message** is used. If **message** is declared in the script but not initialized, the value of **message** will be an empty string. If **message** has not been declared, the script produces an error.

```
<xsl:template match="configuration">
    <xsl:call-template name="print-host-name">
        <xsl:with-param name="message"/>
    </xsl:call-template>
</xsl:template>
```

If you call a template, include the parameter, and define a value for the parameter, the template uses the provided value. The following example calls the named template **print-host-name** with the **message** parameter and a defined value, so the template uses the new value.

```
<xsl:template match="configuration">
    <xsl:call-template name="print-host-name">
        <xsl:with-param name="message"
            select=concat"('Host-name passed in: ', system/host-name)"/>
    </xsl:call-template>
</xsl:template>
```

## Example: Parameters and Match Templates

The following template matches on /, the root of the XML document. It then generates an element named **<outside>**, which is added to the output document, and instructs the Junos OS management process (mgd) to recursively apply templates to the **configuration/system** subtree. The parameter **host** is used in the processing of any matching nodes.The value of the **host** parameter is the value of the **host-name** statement at the [**edit system**] level of the configuration hierarchy.

```
<xsl:template match="/">
  <outside>
    <xsl:apply-templates select="configuration/system">
        <xsl:with-param name="host" select="configuration/system/host-name"/>
    </xsl:apply-templates>
  </outside>
</xsl:template>
```

The following template matches the **<system>** element, which is the top of the subtree selected in the previous example. The **host** parameter is declared with no default value. An **<inside>** element is generated, which contains the value of the **host** parameter that was defined in the **<xsl:with-param>** tag in the previous example.

```
<xsl:template match="system">
  <xsl:param name="host"/>
  <inside>
    <xsl:value-of select="$host"/>
  </inside>
</xsl:template>
```

## Example: Parameters and Named Templates

The following named template **report-changed** declares two parameters: **dot**, which defaults to the current node, and **changed**, which defaults to the **changed** attribute of the node **dot**.

```
<xsl:template name="report-changed">
  <xsl:param name="dot" select="."/>
  <xsl:param name="changed" select="$dot/@changed"/>
  <!-- ... -->
</xsl:template>
```

The next stanza calls the **report-changed** template and defines a source for the **changed** attribute different from the default source defined in the **report-changed** template. When the **report-changed** template is invoked, it will use the newly defined source for the **changed** attribute in place of the default source.

```
<xsl:template match="system">
  <xsl:call-template name="report-changed">
    <xsl:with-param name="changed" select="../@changed"/>
  </xsl:call-template>
</xsl:template>
```

Likewise, the template call can include the **dot** parameter and define a source other than the default current node, as shown here:

```
<xsl:template match="system">
  <xsl:call-template name="report-changed">
```

```
    <xsl:with-param name="dot" select="../../>
  </xsl:call-template>
</xsl:template>
```

**Related Documentation**

- XSLT Templates Overview on page 24
- xsl:param on page 155
- xsl:with-param on page 161

## XSLT Variables Overview

You declare variables using the **<xsl:variable>** element. The **name** attribute specifies the name of the variable, which is case-sensitive. Once you declare a variable, you can reference it within an XPath expression using the variable name prefixed with a dollar sign ($).

Variables are immutable; you can set the value of a variable only when you declare the variable, after which point, the value is fixed. You initialize a variable by including the **select** attribute and an expression in the **<xsl:variable>** tag. The following example declares and initializes the variable **location**. The **location** variable is then used to initialize the **message** variable.

```
<xsl:variable name="location" select="$dot/@location"/>
<xsl:variable name="message" select="concat('We are in ', $location, ' now.')"/>
```

You can define both local and global variables. Variables are global if they are children of the **<xsl:stylesheet>** element. Otherwise, they are local. The value of a global variable is accessible anywhere in the style sheet. The scope of a local variable is limited to the template or code block in which it is defined.

XSLT variables can store any values that you can calculate or statically define. This includes data structures, XML hierarchies, and combinations of text and parameters. For example, you could assign the XML output of an operational mode command to a variable and then access the hierarchy within the variable.

The following template declares the **message** variable. The **message** variable includes both text and parameter values. The template generates a system log message by referring to the value of the message variable. The resulting system log message is as follows:

Device *device-name* was changed on *date* by user '*user*.'

```
<xsl:template name="emit-syslog">
  <xsl:param name="user"/>
  <xsl:param name="date"/>
  <xsl:param name="device"/>
  <xsl:variable name="message">
    <xsl:text>Device </xsl:text>
    <xsl:value-of select="$device"/>
    <xsl:text> was changed on </xsl:text>
    <xsl:value-of select="$date"/>
    <xsl:text> by user '</xsl:text>
    <xsl:value-of select="$user"/>
    <xsl:text>.'</xsl:text>
```

```
    </xsl:variable>
    <syslog>
      <message>
        <xsl:value-of select="$message"/>
      </message>
    </syslog>
  </xsl:template>
```

Table 4 on page 30 provides examples of XSLT variable declarations along with pseudocode explanations.

### Table 4: Examples and Pseudocode for XSLT Variable Declaration

| Variable Declaration | Pseudocode Explanation |
|---|---|
| **<xsl:variable name="mpls" select="protocols/mpls"/>** | Assigns the **[edit protocols mpls]** hierarchy level to the variable named **mpls**. |
| **<xsl:variable name="color" select="data[name = 'color']/value"/>** | Assigns the value of the **color** macro parameter to a variable named **color**. The **<data>** element in the XPath expression is useful in commit script macros. For more information, see "Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements" on page 333. |

**Related Documentation**

- xsl:variable on page 160

## XSLT Programming Instructions Overview

XSLT has a number of traditional programming instructions. Their form tends to be verbose, because their syntax is built from XML elements.

The XSLT programming instructions most commonly used in commit, op, and event scripts, which provide flow control within a script, are described in the following sections:

- <xsl:choose> Programming Instruction on page 30
- <xsl:for-each> Programming Instruction on page 31
- <xsl:if> Programming Instruction on page 31
- Sample XSLT Programming Instructions and Pseudocode on page 32

### <xsl:choose> Programming Instruction

The **<xsl:choose>** instruction is a conditional construct that causes different instructions to be processed in different circumstances. It is similar to a switch statement in traditional programming languages. The **<xsl:choose>** instruction contains one or more **<xsl:when>** elements, each of which tests an XPath expression. If the test evaluates to true, the XSLT processor executes the instructions in the **<xsl:when>** element. After the XSLT processor finds an XPath expression in an **<xsl:when>** element that evaluates to true, the XSLT processor ignores all subsequent **<xsl:when>** elements contained in the **<xsl:choose>** instruction, even if their XPath expressions evaluate to true. In other words, the XSLT processor processes only the instructions contained in the first **<xsl:when>** element

whose **test** attribute evaluates to true. If none of the **<xsl:when>** elements' **test** attributes evaluate to true, the content of the optional **<xsl:otherwise>** element, if one is present, is processed.

The **<xsl:choose>** instruction is similar to a switch statement in other programming languages. The **<xsl:when>** element is the "case" of the switch statement, and you can add any number of **<xsl:when>** elements. The **<xsl:otherwise>** element is the "default" of the switch statement.

```
<xsl:choose>
  <xsl:when test="xpath-expression">
    ...
  </xsl:when>
  <xsl:when test="another-xpath-expression">
    ...
  </xsl:when>
  <xsl:otherwise>
    ...
  </xsl:otherwise>
</xsl:choose>
```

## <xsl:for-each> Programming Instruction

The **<xsl:for-each>** element tells the XSLT processor to gather together a set of nodes and process them one by one. The nodes are selected by the XPath expression specified by the **select** attribute. Each of the nodes is then processed according to the instructions held in the **<xsl:for-each>** construct.

```
<xsl:for-each select="xpath-expression">
  ...
</xsl:for-each>
```

Code inside the **<xsl:for-each>** instruction is evaluated recursively for each node that matches the XPath expression. That is, the current context is moved to each node selected by the **<xsl:for-each>** clause, and processing is relative to that current context.

In the following example, the **<xsl:for-each>** construct recursively processes each node in the **[system syslog file]** hierarchy. It updates the current context to each matching node and prints the value of the **name** element, if one exists, that is a child of the current context.

```
<xsl:for-each select="system/syslog/file">
  <xsl:value-of select="name"/>
</xsl:for-each>
```

## <xsl:if> Programming Instruction

An **<xsl:if>** programming instruction is a conditional construct that causes instructions to be processed if the XPath expression held in the **test** attribute evaluates to **true**.

```
<xsl:if test="xpath-expression">
  ...executed if test expression evaluates to true
</xsl:if>
```

There is no corresponding else clause.

## Sample XSLT Programming Instructions and Pseudocode

Table 5 on page 32 presents examples that use several XSLT programming instructions along with pseudocode explanations.

### Table 5: Examples and Pseudocode for XSLT Programming Instructions

| Programming Instruction | Pseudocode Explanation |
|---|---|
| <xsl:choose><br><xsl:when test="system/host-name"><br><change><br><system><br><host-name>M320</host-name><br></system><br></change><br></xsl:when><br><xsl:otherwise><br><xnm:error><br><message><br>Missing [edit system host-name] M320.<br></message><br></xnm:error><br></xsl:otherwise><br></xsl:choose> | When the **host-name** statement is included at the **[edit system]** hierarchy level, change the hostname to **M320**.<br><br>Otherwise, issue the warning message: **Missing [edit system host-name] M320**. |
| <xsl:for-each select="interfaces/interface[starts-with(name, 'ge-')]/unit"> | For each Gigabit Ethernet interface configured at the **[edit interfaces ge-*fpc/pic/port* unit *logical-unit-number*]** hierarchy level. |
| <xsl:for-each select="data[not(value)]/name"> | Select any macro parameter that does not contain a parameter value.<br><br>In other words, match all **apply-macro** statements of the following form:<br><br>apply-macro *apply-macro-name* {<br>*parameter-name*;<br>}<br><br>And ignore all **apply-macro** statements of the form:<br><br>apply-macro *apply-macro-name* {<br>*parameter-name parameter-value*;<br>} |
| <xsl:if test="not(system/host-name)"> | If the **host-name** statement is not included at the **[edit system]** hierarchy level. |
| <xsl:if test="apply-macro[name = 'no-igp'] | If the **apply-macro** statement named **no-igp** is included at the current hierarchy level. |
| <xsl:if test="not(../apply-macro[name = 'no-ldp']) | If the **apply-macro** statement with the name **no-ldp** is not included two hierarchy levels above the current hierarchy level. |

**Related Documentation**

- xsl:choose on page 150

- xsl:for-each on page 152
- xsl:if on page 153
- xsl:otherwise on page 154
- xsl:when on page 161


## XSLT Recursion Overview

XSLT depends on recursion as a looping mechanism. Recursion occurs when a section of code calls itself, either directly or indirectly. Both named and unnamed templates can use recursion, and different templates can use mutual recursion, one calling another that in turn calls the first.

To avoid infinite recursion and excessive consumption of system resources, the Junos OS management process (mgd) limits the maximum recursion to 5000 levels. If this limit is reached, the script fails.

In the following example, an unnamed template matches on a **<count>** element. It then calls the **<count-to-max>** template, passing the value of the **count** element as **max**. The **<count-to-max>** template starts by declaring both the **max** and **cur** parameters and setting the default value of each to **1** (one). Although the optional default value for **max** is one, the template will use the value passed in from the **count** template. Then the current value of **cur** is emitted in an **<out>** element. Finally, if **cur** is less than **max**, the **<count-to-max>** template recursively invokes itself, passing **cur + 1** as **cur**. This recursive pass then outputs the next number and repeats the recursion until **cur** equals **max**.

```
<xsl:template match="count">
  <xsl:call-template name="count-to-max">
    <xsl:with-param name="max" select="."/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="count-to-max">
  <xsl:param name="cur" select="'1'"/>
  <xsl:param name="max" select="'1'"/>

  <out><xsl:value-of select="$cur"/></out>

  <xsl:if test="$cur &lt; $max">
    <xsl:call-template name="count-to-max">
      <xsl:with-param name="cur" select="$cur + 1"/>
      <xsl:with-param name="max" select="$max"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Given a **max** value of **10**, the values contained in the **<out>** tag are **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, and **10**.

## XSLT Context (Dot) Overview

The current context node changes as an **<xsl:apply-templates>** instruction traverses the document hierarchy and as an **<xsl:for-each>** instruction examines each node that matches an XPath expression. All relative node references are relative to the current context node. This node is abbreviated " **.**" (read: dot) and can be referred to in XPath expressions, allowing explicit references to the current node.

The following example contains four uses for "**.**" . The **system** node is saved in the **system** variable for use inside the **<xsl:for-each>** instruction, where the value of "**.**" will have changed. The **for-each select** expression uses "**.**" to mean the value of the **name** element. The "**.**" is then used to pull the value of the **name** element into the **<tag>** element. The **<xsl:if>** test then uses "**.**" to reference the value of the current context node.

```
<xsl:template match="system">
  <xsl:variable name="system" select="."/>
  <xsl:for-each select="name-server/name[starts-with(., '10.')]">
    <tag><xsl:value-of select="."/></tag>
    <xsl:if test=". = '10.1.1.1'">
      <match>
        <xsl:value-of select="$system/host-name"/>
      </match>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

CHAPTER 5

# Understanding SLAX

This chapter discusses the following topics:



## SLAX Overview

Stylesheet Language Alternative Syntax (SLAX) is a language for writing Junos OS commit scripts, op scripts, and event scripts. It is an alternative to Extensible Stylesheet Language Transformations (XSLT). SLAX has a distinct syntax similar to that of C and Perl, but the same semantics as XSLT.



### SLAX Advantages

XSLT is a powerful and effective tool for handling Extensible Markup Language (XML) that works well for machine-to-machine communication, but its XML-based syntax is inconvenient for the development of complex programs.

SLAX has a simple syntax that follows the style of C and PERL. It provides a practical and succinct way to code, thus allowing you to create readable, maintainable commit, op, and event scripts. SLAX removes XPath expressions and programming instructions

from XML elements. XML angle brackets and quotation marks are replaced by parentheses and curly brackets ({ }), which are the familiar delimiters of C and PERL.

The benefits of SLAX are particularly strong for programmers who are not already accustomed to XSLT, because SLAX allows them to concentrate on the new programming topics introduced by XSLT, rather than concentrating on learning a new syntax. For example, SLAX allows you to:

- Use **if**, **else if**, and **else** statements instead of **<xsl:choose>** and **<xsl:if>** elements
- Put test expressions in parentheses **( )**
- Use the double equal sign (**==**) to test equality instead of the single equal sign (**=**)
- Use curly braces to show containment instead of closing tags
- Perform concatenation using the underscore (_) operator, as in PERL, version 6
- Write text strings using simple quotation marks (**" "**) instead of the **<xsl:text>** element
- Define named templates with a syntax resembling a function definition
- Invoke named templates with a syntax resembling a function call
- Simplify namespace declarations
- Reduce the clutter in your scripts
- Write more readable scripts

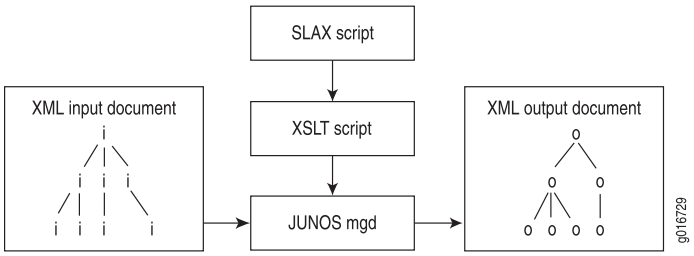
## How SLAX Works

SLAX functions as a preprocessor for XSLT. Junos OS internally translates SLAX programming instructions (such as **if** and **else** statements) into the equivalent XSLT instructions (such as **<xsl:if>** and **<xsl:choose>** elements). After this translation, the XSLT transformation engine—which, for Junos OS, is the Junos OS management (mgd) process—is invoked.

SLAX does not affect the expressiveness of XSLT; it only makes XSLT easier to use. The underlying SLAX constructs are completely native to XSLT. SLAX adds nothing to the XSLT engine. The SLAX parser parses an input document and builds an XML tree identical to the one produced when the XML parser reads an XSLT document.

Figure 2 on page 36 shows the flow of SLAX script input and output.

### Figure 2: SLAX Script Input and Output



**Related Documentation**

- Converting Scripts Between SLAX and XSLT on page 37
- SLAX Elements and Element Attributes Overview on page 42
- SLAX Statements Overview on page 56
- SLAX Syntax Rules Overview on page 39
- SLAX Templates Overview on page 44
- SLAX Variables Overview on page 53
- XPath Expressions Overview for SLAX on page 43
- XSLT Overview on page 19

## Converting Scripts Between SLAX and XSLT

SLAX is a C-like alternative syntax to XSLT and can be viewed as a preprocessor for XSLT. Before Junos OS invokes the XSLT processor, the software converts any SLAX constructs in the script (such as **if/else if/else**) to equivalent XSLT constructs (such as **<xsl:choose>** and **<xsl:if>**). For more information about SLAX, see "SLAX Overview" on page 35.

You can use the **request system scripts convert** operational mode command to convert a script or partial script input written in SLAX or XSLT into the alternate language. Users familiar with C and PERL can convert existing XSLT scripts to SLAX to more easily read and maintain the scripts. In addition, converting a script and studying the results facilitates learning the differences between the two languages.

The following sections explain how to convert a script from one language to the other:

- Converting a Script from SLAX to XSLT on page 37
- Converting a Script from XSLT to SLAX on page 38

### Converting a Script from SLAX to XSLT

To convert a SLAX script to XSLT, issue the **request system scripts convert slax-to-xslt** operational mode command, and specify the source file, the destination directory, and, optionally, a destination file. The source script is the basis for the new script. The source script is not overwritten by the new script.

The command syntax is:

user@host> **request system scripts convert slax-to-xslt source *source/filename* destination *destination/<filename>*** **<partial>**

Starting with Junos OS Release 12.2, you can include the **partial** option to convert partial script input.

The following three examples convert a script from SLAX to XSLT using a source and destination directory relevant to the default storage location for the type of script being converted:

```
user@host> request system scripts convert slax-to-xslt source /var/db/scripts/op/script1.slax
destination /var/db/scripts/op/script1.xsl
conversion complete

user@host> request system scripts convert slax-to-xslt source /var/db/scripts/event/script1.slax
destination /var/db/scripts/event/script1.xsl
conversion complete

user@host> request system scripts convert slax-to-xslt source
/var/db/scripts/commit/script1.slax destination /var/db/scripts/commit/script1.xsl
conversion complete
```

When you issue the **slax-to-xslt** conversion command, the **script1.slax** file remains unchanged in the source directory, and a new script called **script1.xsl** is added to the destination directory.

```
user@host> file list /var/db/scripts/op
script1.slax
script1.xsl
```

If you specify only the destination directory and do not specify a destination filename, the generated filename is **SLAX-Conversion-Temp** or **slax-temp** depending on the Junos OS release, with a randomly generated, five-character, alpha-numeric extension.

```
user@host> request system scripts convert slax-to-xslt source /var/db/scripts/op/script1.slax
destination /var/db/scripts/op/
conversion complete

user@host> file list /var/db/scripts/op
slax-temp.SlhIr
script1.slax
```

## Converting a Script from XSLT to SLAX

To convert an XSLT script to SLAX, issue the **request system scripts convert xslt-to-slax** operational mode command, and specify the source file, the destination directory, and, optionally, a destination file. The source script is the basis for the new script. The source script is not overwritten by the new script.

The command syntax is:

user@host> **request system scripts convert xslt-to-slax source** ***source/filename*** **destination** ***destination/<filename>*** **<partial> <version (1.0 | 1.1)>**

To convert partial script input, include the **partial** option in the command. The **version** option specifies the SLAX version that will be listed in the version statement of the generated script. Specify the version as either 1.0 or 1.1. The default is 1.1. The **partial** and **version** options are supported starting with Junos OS Release 12.2.

The following three examples convert a script from XSLT to SLAX using a source and destination directory relevant to the default storage location for the type of script being converted:

```
user@host> request system scripts convert xslt-to-slax source /var/db/scripts/op/script1.xsl
destination /var/db/scripts/op/script1.slax version 1.0
conversion complete

user@host> request system scripts convert xslt-to-slax source /var/db/scripts/event/script1.xsl
destination /var/db/scripts/event/script1.slax version 1.0
conversion complete

user@host> request system scripts convert xslt-to-slax source /var/db/scripts/commit/script1.xsl
destination /var/db/scripts/commit/script1.slax version 1.0
conversion complete
```

When you issue the **xslt-to-slax** conversion command, the **script1.xsl** file remains unchanged in the source directory, and a new script called **script1.slax** is added to the destination directory.

```
user@host> file list /var/db/scripts/op
script1.slax
script1.xsl
```

The SLAX script boilerplate lists the specified SLAX version. In this example, the version is 1.0.

```
user@host> file show /var/db/scripts/op/script1.slax
/* Machine Crafted with Care (tm) by slaxWriter */
version 1.0;
...
```

If you specify only the destination directory and do not specify a destination filename, the generated filename is **SLAX-Conversion-Temp** or **slax-temp** depending on the Junos OS release, with a randomly generated, five-character, alpha-numeric extension.

```
user@host> request system scripts convert xslt-to-slax source /var/db/scripts/op/script1.xsl
destination /var/db/scripts/op/
conversion complete

user@host> file list /var/db/scripts/op
slax-temp.Vosnd
script1.xsl
```

**Related Documentation**

- SLAX Overview on page 35

## SLAX Syntax Rules Overview

SLAX syntax rules are similar to those of traditional programming languages like C and PERL. The following sections discuss general aspects of SLAX syntax rules:

- Code Blocks on page 40
- Comments on page 40

## Code Blocks

SLAX delimits blocks of code with curly braces. Code blocks, which may define the boundaries of an element, a hierarchy, or a segment of code, can be at the same level as or nested within other code blocks. Declarations defined within a particular code block have a scope that is limited to that block.

The following example shows two blocks of code. Curly braces define the bounds of the **match /** block. The second block, containing the **<op-script-results>** element, is nested within the first.

```
match / {
  <op-script-results> {
      <output> "Script summary:";
  }
}
```

## Comments

In SLAX, you can add comments anywhere in a script. Commenting a script increases readability for all users, including the author, who may need to return to a script long after it was originally written. It is recommended that you add comments throughout a script as you write it.

In SLAX, you insert comments in the traditional C style, beginning with **/*** and ending with ***/**. For example:

```
/* This is a comment. */
```

Multi-line comments follow the same format. In the following example, the additional "*" characters are added to the beginning of the lines for readability, but they are not required.

```
/* Script Title
*  Author: Jane Doe
*  Last modified: 01/01/10
*  Summary of modifications: ...
*/
```

The XSLT equivalent is:

```
<!-- Script Title
 Author: Jane Doe
 Last modified: 01/01/10
 Summary of modifications: ...
-->
```

The following example inserts a comment into the script to remind the programmer that the output is sent to the console.

```
match / {
  <op-script-results> {
    /* Output script summary  to the console */
      <output> "Script summary: ...";
```

```
  }
}
```

## Line Termination

As with many traditional programming languages, SLAX statements are terminated with a semicolon.

In the following example, the namespace declarations, import statement, and output element are all terminated with a semicolon. Lines that begin or end a block are not terminated with a semicolon.

```
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 / {
  <op-script-results> {
    <output> "Script summary:";
     /* ... */
  }
}
```

## Strings

Strings are sequences of text characters. SLAX strings can be enclosed in either single quotes or double quotes. However, you must close the string with the same type of quote used to open the string. Strings can be concatenated together using the SLAX concatenation operation, which is the underscore (_).

For example:

```
match / {
  <op-script-results> {
    /* Output script summary  to the console */
      <output> "Script" _ "summary: ...";
  }
}
```

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42
- SLAX Overview on page 35
- SLAX Statements Overview on page 56SLAX Templates Overview on page 44
- SLAX Templates Overview on page 44
- SLAX Variables Overview on page 53

# SLAX Elements and Element Attributes Overview

## SLAX Elements

SLAX elements are written with only the opening tag. The contents of the tag appear immediately following the opening tag. The contents can be either a simple expression or a more complex expression placed inside curly braces. For example:

```
<top> {
  <one>;
  <two> {
    <three>;
    <four>;
    <five> {
      <six>;
    }
  }
}
```

The XSLT equivalent is:

```
<top>
  <one/>
  <two>
    <three/>
    <four/>
    <five>
      <six/>
    </five>
  </two>
</top>
```

Using these nesting techniques and removing the closing tag reduces clutter and increases code clarity.

## SLAX Element Attributes

SLAX element attributes follow the style of XML. Attributes are included in the opening tag and consist of an attribute name and value pair. The attribute syntax consists of the attribute name followed by an equals sign and then the attribute value enclosed in quotation marks. Multiple attributes are separated by spaces.

```
<element attr1="one" attr2="two">;
```

Where XSLT allows attribute value templates using curly braces, SLAX uses the normal expression syntax. Attribute values can include any XPath syntax, including quoted strings, parameters, variables, numbers, and the SLAX concatenation operator, which is an underscore (_). In the following example, the SLAX element **location** has two attributes, **state** and **zip**:

```
<location state=$location/state zip=$location/zip5 _ "-" _ $location/zip4>;
```

The XSLT equivalent is:

```
<location state="{$location/state}"
        zip="{concat($location/zip5, "-", $location/zip4}"/>
```

In SLAX, curly braces placed inside quote strings are not interpreted as attribute value templates. Instead, they are interpreted as plain-text curly braces.

An escape sequence causes a character to be treated as plain text and not as a special operator. For example, in HTML, an ampersand (&) followed by **lt** causes the less-than symbol (<) to be printed.

In XSLT, the double curly braces (**{{** and **}}**) are escape sequences that cause opening and closing curly braces to be treated as plain text. When a SLAX script is converted to XSLT, the curly braces inside quote strings are converted to double curly braces:

```
<avt sign="{here}">;
```

The XSLT equivalent is:

```
<avt sign="{{here}}"/>
```

**Related Documentation**

- XML Overview on page 11

## XPath Expressions Overview for SLAX

XPath expressions can appear either as the contents of an XML element or as the contents of an **expr** (expression) statement. In either case, the value is translated to either an **<xsl:text>** element, which outputs literal text, or to an **<xsl:value-of>** element, which extracts data from an XML structure.

You encode strings using quotation marks (single or double). The concatenation operator is the underscore (_), as in PERL 6.

In this example, the contents of the **<three>** and **<four>** elements are identical, and the content of the **<five>** element differs only in the use of the XPath **concat()** function. The resulting output is the same in all three cases.

```
<top> {
  <one> "test";
  <two> "The answer is " _ results/answer _ ".";
  <three> results/count _ " attempts made by " _ results/user;
  <four> {
    expr results/count _ " attempts made by " _ results/user;
  }
  <five> {
    expr results/count;
    expr " attempts made by ";
    expr results/user;
  }
  <six> results/message;
}
```

The XSLT equivalent is:

```
<top>
  <one><xsl:text>test</xsl:text></one>
  <two>
    <xsl:value-of select='concat("The answer is ", results/answer, ".")'/>
  </two>
```

```
    <three>
      <xsl:value-of select='concat(results/count, " attempts made by ", results/user)'/>
    </three>
    <four>
      <xsl:value-of select='concat(results/count, " attempts made by ", results/user)'/>
    </four>
    <five>
      <xsl:value-of select="results/count"/>
      <xsl:text> attempts made by </xsl:text>
      <xsl:value-of select="results/user"/>
    </five>
    <six><xsl:value-of select='results/message'/></six>
  </top>
```

**Related Documentation**

- concat( ) on page 143
- SLAX Elements and Element Attributes Overview on page 42
- SLAX Syntax Rules Overview on page 39
- XPath Overview on page 22
- xsl:text on page 159
- xsl:value-of on page 159

## SLAX Templates Overview

A SLAX script consists of one or more sets of rules called *templates*. Each template is a segment of code that contains rules to apply when a specified node is matched.

There are two types of templates, named and unnamed (or match), and they are described in the following sections.

- Unnamed (Match) Templates on page 44
- Named Templates on page 45

### Unnamed (Match) Templates

Unnamed templates, also known as match templates, contain a **match** statement with an XPath expression to specify the criteria for nodes upon which the template should be invoked. In the following commit script sample, the template matches the top-level element in the configuration hierarchy.

```
match configuration {
  /* ...body of the  template goes here */
}
```

By default, the processor recursively traverses the entire document hierarchy, inspecting each node, looking for a template that matches the current node. When a matching template is found, the contents of that template are evaluated.

The **apply-templates** statement can be used inside an unnamed template to limit and control the default, hierarchical traversal of nodes. This statement accepts an optional XPath expression, which is equivalent to the **select** attribute in an **<xsl:apply-templates>**

element. If an optional XPath expression is included, only nodes matching the XPath expression are traversed. Otherwise all children of the context node are traversed. If the XPath expression is included but does not match any nodes, nothing is traversed and nothing happens.

In the following example, the template rule matches the **<route>** element in the XML hierarchy. All the nodes containing a **changed** attribute are processed. All **route** elements containing a **changed** attribute are replaced with a **new** element.

```
match route {
  <new> {
    apply-templates *[@changed];
  }
}
```

The XSLT equivalent:

```
<xsl:template match="route">
  <new>
    <xsl:apply-templates select="*[@changed]"/>
  </new>
</xsl:template>
```

Using unnamed templates allows the script to ignore the location of a tag in the XML hierarchy. For example, if you want to convert all **<author>** tags into ** ** tags, using templates enables you to write a single rule that converts all **<author>** tags, regardless of their location in the input XML document.

## Named Templates

Named templates operate like functions in traditional programming languages. When the complexity of a script increases or a code segment appears in multiple places, you can modularize the code and create named templates. Like functions, named templates accept arguments and run only when explicitly called.

In SLAX, the named template definition consists of the **template** keyword, the template name, a set of parameters, and a braces-delimited block of code. Parameter declarations can be inline and consist of the parameter name, and, optionally, a default value. Alternatively, you can declare parameters inside the template block using the **param** statement. If a default value is not defined, the parameter defaults to an empty string.

The following example creates a template named **my-template** and defines three parameters, one of which defaults to the string **false**, and one of which defaults to the contents of the element node named **name** that is a child of the current context node. If the script calls the template and does not pass in a parameter, the default value is used.

```
template my-template ($a, $b = "false", $c = name) {
  /* ... body of the template ... */
}
```

An alternate method is to declare the parameters within the template using the **param** statement. The following code is identical to the previous example:

```
template my-template {
  param $a;
  param $b = "false";
```

```
  param $c = name;
  /* ... body of the template ... */
}
```

In SLAX, you invoke named templates using the **call** statement, which consists of the **call** keyword and template name followed by a set of parameter bindings. These bindings are a comma-separated list of parameter names that are passed into the template from the calling environment. Parameter assignments are made by name and not by position in the list. Alternatively, you can declare parameters inside the **call** block using the **with** statement. Parameters passed into a template must match a parameter defined in the actual template; otherwise the parameter is ignored. Optionally, you can set a value for each parameter. If you do not define a value for the parameter in the calling environment, the script passes in the current value of the parameter if it was previously initialized, or it generates an error if the parameter was never declared. For more information about passing parameters, see "SLAX Parameters Overview" on page 49.

In the following example, the template **my-template** is called with the parameter **c** containing the contents of the element node named **other-name** that is a child of the current context node.

```
call my-template {
  with $c = other-name;
}
```

In the following example, the **name-servers-template** declares two parameters **name-servers** and **size**. The **size** parameter is given a default value of zero. The match template, which declares and initializes **name-servers**, calls the **name-servers-template** three times.

The first call to the template does not include any parameters. Thus **name-servers** will default to an empty string, and **size** will default to a value of zero as defined in the template. The second call includes the **name-servers** and **size** parameters but only supplies a value for the **size** parameter. Thus **name-servers** has the value defined by its initialization in the script, and **size** is equal to the number of **name-servers** elements in the configuration hierarchy. The last call is identical to the second call, but it supplies the parameters using the **with** statement syntax.

```
match configuration {
  param $name-servers = name-servers/name;
  call name-servers-template();
  call name-servers-template($name-servers, $size = count($name-servers));
  call name-servers-template() {
    with $name-servers;
    with $size = count($name-servers);
  }
}
  template name-servers-template($name-servers, $size = 0) {
    <output> "template called with size " _ $size;
  }
```

The XSLT equivalent is:

```
<xsl:template match="configuration">
  <xsl:variable name="name-servers" select="name-servers/name"/>
  <xsl:call-template name="name-servers-template"/>
  <xsl:call-template name="name-servers-template">
```

```
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
  <xsl:call-template name="name-servers-template">
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="name-servers-template">
  <xsl:param name="name-servers"/>
  <xsl:param name="size" select="0"/>
  <output>
    <xsl:value-of select="concat('template called with size ', $size)"/>
  </output>
</xsl:template>
```

**Related Documentation**

- SLAX Parameters Overview on page 49
- XSLT Templates Overview on page 24
- apply-templates on page 165
- call on page 169
- match on page 185
- param on page 195
- with on page 209

## SLAX Functions Overview

Version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases, supports functions. When the complexity of a script increases or a code segment appears in multiple places, you can modularize the code and create functions. Functions accept arguments and run only when explicitly called.

Functions have several advantages over templates, including the following:

- Arguments are passed by position rather than name.
- Return values can be objects as opposed to result tree fragments.
- Functions can be used in expressions.
- Functions can be resolved dynamically (using EXSLT **dyn:evaluate()**).

In SLAX, you define a function definition as a top-level statement in the script. The function definition consists of the **function** keyword, the function name, a set of arguments, and a braces-delimited block of code. The function name must be a qualified name. The argument list is a comma-separated list of parameter names, which are positionally assigned based on the function call. Trailing arguments can have default values. Alternatively, you can define function parameters inside the function block using the **param** statement. The syntax is:

```
function function-name (argument-list) {
  ...
  result return-value;
}

function function-name () {
  param param-name1;
  param param-name2;
  param param-name3 = default-value;
  ...
  result return-value;
}
```

The return value can be a scalar value, an XML element or XPath expression, or a set of instructions that emit the value to be returned.

If there are fewer arguments in the function invocation than in the definition, the default values are used for any trailing arguments. If there are more arguments in the function invocation than in the definition, the function call generates an error.

The following example defines the function **size**, which has three parameters: **width**, **height**, and **scale**. The default value for **scale** is 1. If the function call argument list does not include the **scale** argument, the calculation uses the default value of 1 for that argument. The function's return value is the product of the **width**, **height**, and **scale** variables enclosed in a **<size>** element.

In the main match template, the function call uses width and height data selected from each **graphic/dimension** element in the source XML file. The script evaluates the function, and the **copy-of** statement emits the return value to the result tree as the contents of the **<out>** element.

```
version 1.1;
ns my = "http://www.example.com/myfunctions";

function my:size ($width, $height, $scale = 1) {
  result <size> {
    expr $width * $height * $scale;
  }
}

match / {
  for-each (graphic/dimension) {
    <out> {
      copy-of my:size((width/.), (height/.));
    }
  }
}
```

The following function definition uses **param** statements to define the parameters instead of a comma-separated list. The behavior of the function is identical to that in the previous example.

```
version 1.1;
ns my = "http://www.example.com/myfunctions";

function my:size () {
  param $width;
```

```
  param $height;
  param $scale = 1;
  result <size> {
    expr $width * $height * $scale;
  }
}

match / {
  for-each (graphic/dimension) {
    <out> {
      copy-of my:size((width/.), (height/.));
    }
  }
}
```

Related Documentation

- function on page 180
- param on page 195
- result on page 199

## SLAX Parameters Overview

Parameters can be passed to named or unnamed templates or to functions. After declaring a parameter, you can reference it by prefixing the parameter name with the dollar sign ($).



### Declaring Parameters

In SLAX scripts, you declare parameters using the **param** statement. Optionally, you can define an initial value for each parameter in the declaration. For example:

```
param $dot = .;
```

The scope of a parameter can be local or global. Local parameters must be declared at the beginning of a block, and their scope is limited to the block in which they are declared. A parameter whose value is set by Junos OS at script initialization must be defined as a global parameter. Global parameter declarations are placed just after the style sheet declarations. A script can assign a default value to the global parameter, which is used in the event that Junos OS does not give a value to the parameter.

```
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";
ns ext = "http://xmlsoft.org/XSLT/namespace";
```

```
/* global parameter */
param $interface1 = “fxp0”;
```

In a template, you declare parameters either in a parameter list or by using the **param** statement in the template block. Optionally, declare default values for each template parameter. If a template is invoked without the parameter, the default expression is evaluated, and the results are assigned to the parameter. If you do not define a default value in the template, the parameter defaults to an empty string.

The following named template **print-host-name** declares the parameter **message** and defines a default value:

```
template print-host-name ($message = "host name: " _ system/host-name) {
  <xnm:warning> {
    <message> $message;
  }
}
```

An alternative, but equivalent, declaration is:

```
template print-host-name () {
  param $message = "host name: " _ system/host-name;
  <xnm:warning> {
    <message> $message;
  }
}
```

The template declares **message** and accesses its value by prefixing the parameter name with the dollar sign ($). In XSLT, the parameter name is prefixed by the dollar sign when you access it but not when you declare it.

In a function, you declare parameters either in a parameter list or by using the **param** statement in the function block. Optionally, you can declare default values for trailing parameters. If you invoke a function without that trailing parameter, the default expression is evaluated, and the results are assigned to the parameter. If you do not define a default value, the parameter defaults to an empty string.

The following example defines a function named **size**, which has three parameters: **width**, **height**, and **scale**. The default value for **scale** is 1. If the function call argument list does not include the **scale** argument, the calculation uses the default value of 1 for that argument. The return value for the function is the product of the **width**, **height**, and **scale** variables enclosed in a **<size>** element.

```
function my:size ($width, $height, $scale = 1) {
  result <size> {
    expr $width * $height * $scale;
  }
}
```

An alternative, but equivalent declaration, which uses the **param** statement, is:

```
function my:size () {
  param $width;
  param $height;
  param $scale = 1;
  result <size> {
    expr $width * $height * $scale;
```

```
  }
}
```

## Passing Parameters to Templates

When you invoke a template, you pass arguments into the template either in an argument list or by using the **with** statement. The name of the parameter supplied in the calling environment must match the name of a parameter defined in the actual template. Otherwise, the parameter is ignored. Optionally, for each parameter you pass to a template, you can define a value using an equal sign (=) and a value expression. In the following example, the two calls to the named template **print-host-name** are identical:

```
match configuration {
   call print-host-name($message = "passing in host name: " _ system/host-name);
}
match configuration {
   call print-host-name() {
      with $message = "passing in host name: " _ system/host-name;
   }
}
```

The parameter value that gets used in a template depends on how the template is called. The following three examples, which call the **print-host-name** template, illustrate the possible calling environments.

If you call a template but do not include a specific parameter, the default expression defined in the template is evaluated, and the results are assigned to the parameter. If there is no default value for that parameter in the template, the parameter defaults to an empty string. The following example calls the named template **print-host-name** but does not include any parameters in the call. In this case, the named template will use the default value for the **message** parameter that was defined in the **print-host-name** template, or an empty string if no default exists.

```
match configuration {
   call print-host-name();
}
```

If you call a template and include a parameter, but do not define a value for the parameter in the calling environment, the script passes in the current value of the parameter if it was previously initialized, or it generates an error if the parameter was never declared. The following example calls the named template **print-host-name** and passes in the **message** parameter but does not include a value. If **message** is declared and initialized in the script, and the scope is visible to the block, the current value of **message** is used. If **message** is declared in the script but not initialized, the value of **message** will be an empty string. If **message** has not been declared, the script produces an error.

```
match configuration {
   call print-host-name($message);
   /* If $message was initialized previously, the current value is used;
    * If $message was declared but not initialized, an empty string is used;
    * If $message was never declared, the call generates an error. */
}
```

If you call a template, include the parameter, and define a value for the parameter, the template uses the provided value. The following example calls the named template **print-host-name** with the **message** parameter and a defined value, so the template uses the new value:

```
match configuration {
   call print-host-name($message = "passing in host name: " _ system/host-name);
}
```

## Example: Parameters and Match Templates

The following example matches the top level **configuration** hierarchy element and then instructs the Junos OS management process (mgd) to recursively apply templates to the **system/host-name** subtree. The parameters **message** and **domain** are used in the processing of any matching nodes.

```
match configuration {
  var $domain = domain-name;
  apply-templates system/host-name {
    with $message = "Invalid host-name";
    with $domain;
  }
}

match host-name {
  param $message = "Error";
  param $domain;
  <hello> $message _ ":: " _ . _ " (" _ $domain _ ")";
}
```

The XSLT equivalent is:

```
<xsl:template match="configuration">
  <xsl:apply-templates select="system/host-name">
    <xsl:with-param name="message" select="'Invalid host-name'"/>
    <xsl:with-param name="domain" select="$domain"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="host-name">
  <xsl:param name="message" select="'Error'"/>
  <xsl:param name="domain"/>
  <hello>
    <xsl:value-of select="concat($message, ':: ', .., ' (', $domain, ')')"/>
  </hello>
</xsl:template>
```

## Passing Parameters to Functions

Version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases, supports functions. Although you can use the **param** statement to define function parameters, you cannot use the **with** statement to pass parameter values into the function from the calling environment. When you call a function, you pass arguments into the function in a comma-separated list. Function arguments are passed to the function by position rather than by name as in a template.

A function declaration can define default values for trailing arguments. If there are fewer arguments in the function invocation than in the definition, the default values are used for any trailing arguments. If there are more arguments in the function invocation than in the definition, the function call generates an error.

In the following match template, the function call uses width and height data selected from each **graphic/dimension** element in the source XML file. The script evaluates the function, and the **copy-of** statement emits the return value to the result tree as the contents of the **<out>** element. The function call includes arguments for **width** and **height**, but not for **scale**. The default value of 1 is used for **scale** within the function block.

```
version 1.1;
ns my = "http://www.example.com/myfunctions";

function my:size () {
  param $width;
  param $height;
  param $scale = 1;
  result <size> {
    expr $width * $height * $scale;
  }
}

match / {
  for-each (graphic/dimension) {
    <out> {
      copy-of my:size((width/.), (height/.));
    }
  }
}
```

**Related Documentation**

- SLAX Functions Overview on page 47
- SLAX Templates Overview on page 44
- function on page 180
- param on page 195
- template on page 202
- with on page 209

## SLAX Variables Overview

SLAX variables can store any values that you can calculate or statically define. This includes data structures, XML hierarchies, and combinations of text and parameters. For example, you could assign the XML output of an operational mode command to a variable and then access the hierarchy within the variable.

You can define both local and global variables. Variables are global if they are defined outside of any template. Otherwise, they are local. The value of a global variable is accessible anywhere in the script. The scope of a local variable is limited to the template or code block in which it is defined.

Version 1.0 of the SLAX language supports immutable variables, which are declared using the **var** statement. Version 1.1 of the SLAX language, which is supported starting with Junos OS Release 12.2, introduces mutable variables, which are declared using the **mvar** statement. Mutable and immutable variables are discussed in the following sections:


- Immutable variables on page 54
- Mutable variables on page 55


## Immutable variables

In version 1.0 of the SLAX language, you declare variables using the **var** statement. Variables declared using the **var** statement are immutable. You can set the value of an immutable variable only when you declare it, after which point the value is fixed.

In the declaration, the variable name is prefixed with the dollar sign ($), which is unlike the XSLT declaration, where the dollar sign does not prefix the value of the **name** attribute of the **<xsl:variable>** element. Once you declare a variable, you can reference it within an XPath expression using the variable name prefixed with a dollar sign ($). You initialize a variable by following the variable name with an equal sign (=) and an expression.

The following example declares and initializes the variable **location**, which is then used to initialize the variable **message**:

```
var $location = $dot/@location;
var $message = "We are in " _ $location _ " now.";
```

The XSLT equivalent is:

```
<xsl:variable name="location" select="$dot/@location"/>
<xsl:variable name="message" select="concat('We are in ', $location, ' now.')"/>
```

Variables declared using the **var** statement are immutable. As such, you can never change the value of the variable after it is initialized in the declaration. Although you cannot directly update the value of the variable, you can mimic the effect by recursively calling a function and passing in the value of the variable as a parameter. For example:

```
var $count = 1;
match / {
  call update-count($myparam = $count);
}
template update-count($myparam) {
  expr $count _ ", " $myparam _"\n";
  if ($myparam != 4) {
    call update-count($myparam = $myparam + 1)
  }
}
```

Executing the op script in the CLI produces the following output in the log file. Although the **count** variable must remain fixed, **myparam** is updated with each call to the template.

```
1, 1
1, 2
1, 3
1, 4
1, 5
```

## Mutable variables

Version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases, introduces mutable variables. Unlike variables declared using the **var** statement, the value of a mutable variable can be modified by a script. You can set the initial value of a mutable variable at the time you declare it or at any point in the script.

You declare mutable variables using the **mvar** statement. In the declaration, the variable name is prefixed with the dollar sign ($). Once you declare a mutable variable, you can reference it within an XPath expression using the variable name prefixed with a dollar sign ($). You initialize the variable by following the variable name with an equal sign (=) and an expression.

The following example declares and initializes the mutable variable **location**, which is then used to initialize the mutable variable **message**:

```
mvar $location = $dot/@location;
mvar $message = "We are in " _ $location _ " now.";
```

Mutable variables can be initialized or updated after they are declared. To initialize or update the value of a mutable variable, use the **set** statement. The following example declares the variable, **block**, and initializes it with the element **<block>**:

```
mvar $block;
set $block = <block> "start here";
```

For mutable variables that represent a node set, use the **append** statement to append a new node to the existing node set. The following example creates the mutable variable **$mylist**, which is initialized with one **<item>** element. For each grocery item in the **$list** variable, the script appends an **<item>** element to the **$mylist** node set with the name and size of the item.

```
version 1.1;

var $list := {
 <list> {
   <grocery> {
     <name> "milk";
     <type> "nonfat";
     <brand> "any";
     <size> "gallon";
   }
   <grocery> {
     <name> "orange juice";
     <type> "no pulp";
     <brand> "any";
     <size> "half gallon";
   }
   <drugstore>{
     <name> "aspirin";
     <brand> "any";
     <size> "50 tablets";
   }
 }
}
```

```
match / {

  mvar $mylist;
  set $mylist = <item> {
    <name> "coffee";
    <size> "1 lb";
  }
  for $item ($list/list/grocery) {
    append $mylist += <item> {
      <name> $item/name;
      <size> $item/size;
    }
  }
  <grocery-short-list> {
    copy-of $mylist;
  }
}
```

The output from the script is:

```
<?xml version="1.0"?>
<grocery-short-list>
  <item>
    <name>coffee</name>
    <size>1 lb</size>
  </item>
  <item>
    <name>milk</name>
    <size>gallon</size>
  </item>
  <item>
    <name>orange juice</name>
    <size>half gallon</size>
  </item>
</grocery-short-list>
```

**Related Documentation**

- XSLT Variables Overview on page 29
- SLAX Parameters Overview on page 49
- append on page 164
- mvar on page 187
- set on page 200
- var on page 207

## SLAX Statements Overview

This section lists some commonly used SLAX statements, with brief examples and XSLT equivalents.

- for-each Statement on page 57
- if, else if, and else Statements on page 58

- match Statement on page 59
- ns Statement on page 59
- version Statement on page 60


## for-each Statement

The SLAX **for-each** statement functions like the **<xsl:for-each>** element. The statement consists of the **for-each** keyword, a parentheses-delimited expression, and a curly braces-delimited block. The **for-each** statement tells the processor to gather together a set of nodes and process them one by one. The nodes are selected by the specified XPath expression. Each of the nodes is then processed according to the instructions held in the **for-each** code block.

```
for-each (xpath-expression) {
  ...
}
```

Code inside the **for-each** instruction is evaluated recursively for each node that matches the XPath expression. That is, the current context is moved to each node selected by the **for-each** clause, and processing is relative to that current context.

In the following example, the **inventory** variable stores the inventory hierarchy. The **for-each** statement recursively processes each **chassis-sub-module** node that is a child of **chassis-module** that is a child of the **chassis** node. For each **chassis-sub-module** element that contains a **part-number** with a value equal to the specified part number, a **message** element is created that includes the name of the chassis module and the name and description of the chassis sub module.

```
for-each ($inventory/chassis/chassis-module/
        chassis-sub-module[part-number == '750-000610']) {
  <message> "Down rev PIC in " _ ../name _ ", " _ name _ ": " _ description;
}
```

The XSLT equivalent is:

```
<xsl:for-each select="$inventory/chassis/chassis-module/
          chassis-sub-module[part-number = '750-000610']">
  <message>
    <xsl:value-of select="concat('Down rev PIC in ', ../name, ', ', name, ': ',
          description)"/>
  </message>
</xsl:for-each>
```

## if, else if, and else Statements

SLAX supports **if**, **else if**, and **else** statements. The **if** statement is a conditional construct that causes instructions to be processed if the specified XPath expression evaluates to true. The **if** construct may have one or more associated **else if** clauses, each of which tests an XPath expression. If the expression in the **if** statement evaluates to false, the processor checks each **else if** expression. If a statement evaluates to true, the script executes the instructions in the associated block and ignores all subsequent **else if** and **else** statements. The optional **else** clause is the default code that is executed in the event that all associated **if** and **else-if** expressions evaluate to false. If all of the **if** and **else if** statements evaluate to false, and the **else** statement is not present, no action is taken.

The expressions that appear in parentheses are extended XPath expressions, which support the double equal sign (==) in place of XPath's single equal sign (=).

```
if (expression) {
  /* If block Statement */
}
else if (expression) {
   /* else if block statement */
}
else {
  /* else block statement */
}
```

During script processing, an **if** statement that does not have an associated **else if** or **else** statement is transformed into an **<xsl:if>** element. If either the **else if** or **else** clauses are present, the **if** statement and associated **else if** and **else** blocks are transformed into an **<xsl:choose>** element.

```
if (starts-with(name, "fe-")) {
  if (mtu < 1500) {
    /* Select Fast Ethernet interfaces with low MTUs */
  }
}
else {
  if (mtu > 8096) {
    /* Select non-Fast Ethernet interfaces with high MTUs */
  }
}
```

The XSLT equivalent is:

```
<xsl:choose>
  <xsl:when test="starts-with(name, 'fe-')">
    <xsl:if test="mtu &lt; 1500">
      <!-- Select Fast Ethernet interfaces with low MTUs -->
    </xsl:if>
  </xsl:when>
  <xsl:otherwise>
    <xsl:if test="mtu &gt; 8096">
      <!-- Select non-Fast Ethernet interfaces with high MTUs -->
    </xsl:if>
  </xsl:otherwise>
</xsl:choose>
```

## match Statement

You specify basic match templates using the **match** statement, followed by an expression specifying when the template should be allowed and a block of statements enclosed in a set of braces.

```
match configuration {
   <xnm:error> {
     <message> "...";
  }
}
```

The XSLT equivalent is:

```
<xsl:template match="configuration">
  <xnm:error>
    <message> ...</message>
  </xnm:error>
</xsl:template>
```

For more information about constructing match templates, see "SLAX Templates Overview" on page 44.

## ns Statement

You specify namespace definitions using the SLAX **ns** statement. This consists of the **ns** keyword, a prefix string, an equal sign, and a namespace Uniform Resource Identifier (URI). To define the default namespace, use only the **ns** keyword and a namespace URI.

```
ns junos = "http://www.juniper.net/junos/";
```

The **ns** statement can appear after the **version** statement at the beginning of the style sheet or at the beginning of any block.

```
ns a = "http://example.com/1";
ns "http://example.com/global";
ns b = "http://example.com/2";
match / {
  ns c = "http://example.com/3";
  <top> {
    ns a = "http://example.com/4";
    apply-templates commit-script-input/configuration;
  }
}
```

When it appears at the beginning of the style sheet, the **ns** statement can include either the **exclude** or **extension** keyword. The keyword instructs the parser to add the namespace prefix to the **exclude-result-prefixes** or **extension-element-prefixes** attribute.

```
ns exclude foo = "http://example.com/foo";
ns extension jcs = "http://xml.juniper.net/jcs";
```

The XSLT equivalent is:

```
<xsl:stylesheet xmlns:foo="http://example.com/foo"
  xmlns:jcs="http://xml.juniper.net/jcs"
  exclude-result-prefixes="foo"
  extension-element-prefixes="jcs">
```

```
  <!-- ... -->
</xsl:stylesheet>
```

### version Statement

All SLAX style sheets must begin with a **version** statement, which specifies the version number for the SLAX language. Supported versions include 1.0 and 1.1. SLAX version 1.0 uses XML version 1.0 and XSLT version 1.1.

```
version 1.0;
```

The XSLT equivalent is:

```
<xsl:stylesheet version="1.0">
```

**Related Documentation**

- else on page 173
- for-each on page 178
- if on page 181
- match on page 185
- version on page 207

## XSLT Elements Without SLAX Equivalents

Some XSLT elements are not directly translated into SLAX statements. Some examples of XSLT elements for which there are no SLAX equivalents in SLAX version 1.0 are **<xsl:fallback>**, **<xsl:output>**, and **<xsl:sort>**.

You can encode these elements directly as normal SLAX elements in the XSLT namespace. For example, you can include the **<xsl:output>** and **<xsl:sort>** elements in a SLAX script, as shown here:

```
<xsl:output method="xml" indent="yes" media-type="image/svg">;
match * {
  for-each (configuration/interfaces/unit) {
    <xsl:sort order="ascending">;
  }
}
```

When you include XSLT namespace elements in a SLAX script, do not include closing tags. For empty tags, do not include a forward slash (/) after the tag name. The examples shown in this section demonstrate the correct syntax.

The following XSLT snippet contains a combination of elements, some of which have SLAX counterparts and some of which do not:

```
<xsl:loop select="title">
  <xsl:fallback>
    <xsl:for-each select="title">
      <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:fallback>
</xsl:loop>
```

The SLAX conversion uses the XSLT namespace for XSLT elements that do not have SLAX counterparts:

```
<xsl:loop select = "title"> {
  <xsl:fallback> {
    for-each (title) {
      expr .;
    }
  }
}
```

## SLAX Operators

SLAX provides a variety of operators, which add great versatility to the SLAX scripting language. Table 6 on page 61 summarizes the available operators and provides an example and an explanation of each.

### Table 6: SLAX Operators

| Name | Operator | Example / Explanation |
|---|---|---|
| Addition | + | var $result = 1 + 1;<br><br>Return the sum of the operands. This example assigns a value of 2 to the **$result** variable. |
| And | && | ($byte-count > 500000) && ($byte-count < 1000000)<br><br>Evaluate two expressions and return one boolean result. If either of the two expressions evaluates to **false**, then the combined expression evaluates to **false**. |
| Assignment | = | var $mtu = 1500;<br>mvar $mtu2 = 48;<br>set $mtu2 = 1500;<br><br>Assign a value to a variable or parameter or assign a namespace to a prefix. The example assigns a value of 1500 to both the **$mtu** variable and the **$mtu2** mutable variable. **$mtu2** was originally initialized with a value of 48. |
| Conditional | ?: | var $result = ($a < 10) ? $b : $c;<br><br>Provide conditional assignment based on the boolean value of the evaluated condition. If the conditional expression evaluates to **true**, the entire expression assumes the value of the operand to the left of the colon. If the conditional expression evaluates to **false**, the entire expression assumes the value of the operand to the right of the colon. This operator was introduced in version 1.1 of the SLAX language, which is supported starting with Junos OS Release 12.2.<br><br>In the example, if the value stored in the variable **$a** is less than 10, **$result** is assigned the value stored in **$b**. Otherwise, **$result** is assigned the value stored in **$c**. |
| Division | div | <output>$bit-count div 8;<br><br>Return the result of dividing the left operand by the right operand. If the remainder of the division is nonzero, the result is expressed in decimal floating-point notation. The example divides the **$bit-count** variable by eight, returning the byte count (requires that **$bit-count** has been initialized). |

Table 6: SLAX Operators *(continued)*

| Name | Operator | Example / Explanation |
|---|---|---|
| Equality | == | $mtu == 1500<br><br>Return **true** if the values of the left and right operands are equal; otherwise, the expression returns **false**. In the example, if **$mtu** equals 1500, then the expression resolves to **true**; otherwise, it returns false (requires that **$mtu** has been initialized). |
| Greater than | > | $hop-count > 0<br><br>Return **true** if the value of the left operand is greater than the value of the right operand; otherwise, the expression returns **false**. In this example, if **$hop-count** is greater than zero, the expression returns **true** (requires that **$hop-count** has been initialized). |
| Greater than or equal to | >= | $hop-count >= 1<br><br>Return **true** if the value of the left operand is either greater than or equal to the value of the right operand; otherwise, the expression returns **false**. In this example, if **$hop-count** is 1 or greater, the expression returns **true** (requires that **$hop-count** has been initialized). |
| Inequality | != | $mtu != 1500<br><br>Return **true** if the values of the left and right operands are not equal; otherwise, the expression returns **false**. In the example, if **$mtu** does not equal 1500, then the expression resolves to **true**; otherwise, the expression returns **false** (requires that **$mtu** has been initialized) |
| Iteration | ... | for $i (1 ... 10) {<br>  <player number=$i>;<br>}<br><br>Iterate through a range of integer values with a start value equal to the left operand and an end value equal to the right operand. If the left operand is greater than the right, the numbers are generated in decreasing order. The operator translates into an XPath function that generates the sequence as a node set. This operator was introduced in version 1.1 of the SLAX language, which is supported starting with Junos OS Release 12.2. |
| Less than | < | $hop-count < 15<br><br>Return **true** if the value of the left operand is less than the value of the right operand; otherwise, the expression returns **false**. In this example, if **$hop-count** is less than 15, the expression returns **true** (requires that **$hop-count** has been initialized). |
| Less than or equal to | <= | $hop-count <= 14<br><br>Return **true** if the value of the left operand is either less than or equal to the value of the right operand; otherwise, the expression returns **false**. In this example, if **$hop-count** is 14 or less, the expression returns **true** (requires that **$hop-count** has been initialized). |
| Modulo | mod | <output> 10 mod 3;<br><br>Return the division remainder of two numbers. In this example, the expression outputs a value of 1. |
| Multiplication | * | <output> 5 * 10;<br><br>Return the product of the operands. In this example, the expression outputs a value of 50. |

Table 6: SLAX Operators *(continued)*

| Name | Operator | Example / Explanation |
|---|---|---|
| Node Set, append to | += | mvar $block = <block> "start here";<br>append $block += <block> "next block";<br><br>Append a value to a node set contained in a mutable variable, which is defined with the **mvar** statement. This operator was introduced in version 1.1 of the SLAX language, which is supported starting with Junos OS Release 12.2. |
| Node Set Conversion | := | var $new-node-set := $rtf-variable;<br><br>Convert a result tree fragment into a node set. A result tree fragment contains an unparsed XML data structure. It is not possible to retrieve any of the embedded XML information from this data type. A script can convert the result tree fragment into a node set and then search the node set for the appropriate information and extract it. This operator is supported in Junos OS Release 9.2 and later releases. |
| Or | \|\| | ($mtu-size != 1500) \|\| ($mtu-size > 2000)<br><br>Evaluate two expressions and return one boolean result. If either of the two expressions evaluates to true, then the combined expression evaluates to true. |
| Parentheses | ( ) | var $result = ( $byte-count * 8 ) + 150;<br><br>Create complex expressions. Parentheses function the same way as in a mathematical expression, where the expression within the parentheses is evaluated first. Parentheses can be nested; the innermost set of parentheses is evaluated first, then the next set, and so on. |
| String concatenation | _ (under-score) | var $combined-string = $host-name _ " is located at " _ $location;<br><br>Concatenate multiple strings (note that strings cannot be combined using the + operator in SLAX). In the example, if **$host-name** is "r1" and **$location** is "HQ", then the value of **$combined-string** is "r1 is located at HQ". |
| Subtraction | - | var $result = 64 - 14;<br><br>Return the difference between the left operand and the right operand. This example assigns a value of 50 to the **$result** variable. |
| Unary Minus | - | mvar $number = 5.<br>set $number = - $number;<br><br>Negate the value of the operand, changing a positive value to a negative value or a negative value to a positive value. The example negates the value stored in **$number** and reassigns the new value of -5 to the variable. |
| Union | \| | var $all-interface-nodes = $fe-interface-nodes \| $ge-interface-nodes;<br><br>Create a union of two node sets. All the nodes from one set combine with the nodes in the second set. This is useful when a script needs to perform a similar operation over XML nodes that are pulled from multiple sources. |

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42
- SLAX Overview on page 35

- SLAX Statements Overview on page 56
- SLAX Syntax Rules Overview on page 39
- SLAX Variables Overview on page 53

CHAPTER 6

# Understanding libslax



## libslax Distribution Overview

SLAX is an alternative syntax for XSLT and is tailored for readability and familiarity, following the style of C and Perl. In the SLAX language, programming constructs and XPath expressions are moved from the XML elements and attributes used in XSLT to first class language constructs. SLAX was originally developed as part of Junos OS for the purpose of on-box scripting to allow users to customize and enhance the command-line interface (CLI).

libslax is an open-source implementation of the SLAX language using the "New BSD License". libslax is written in C and is built on top of the libxml2, libxslt, and libexslt libraries. The libslax distribution contains the libslax library, which incorporates a SLAX writer and SLAX parser, a debugger, a profiler, and the SLAX processor (slaxproc). The SLAX processor is a command-line tool that can validate SLAX script syntax, convert between SLAX and XSLT formats, and format, debug, or run SLAX scripts.

The libslax tools are included as part of the standard Junos OS. However, you can download and install the libslax distribution on a computer with a UNIX-like operating system to develop SLAX scripts outside of Junos OS. Current releases, source code, additional documentation, and support materials for libslax are available at:

http://code.google.com/p/libslax/

The SLAX community and support site is available at:

http://www.libslax.org/

**Related Documentation**

- Downloading and Installing the libslax Distribution on page 66
- SLAX Overview on page 35
- Understanding the SLAX Processor (slaxproc) on page 80
- libslax Library and Extension Libraries Overview on page 68
- Using the SLAX Processor (slaxproc) on page 84

## Downloading and Installing the libslax Distribution

The libslax distribution contains the libslax library, which incorporates a SLAX writer and SLAX parser, a debugger, a profiler, and the SLAX processor (slaxproc). The SLAX processor is a command-line tool that can validate SLAX script syntax, convert between SLAX and XSLT formats, and format, debug, or run SLAX scripts.

The libslax tools are included as part of the standard Junos OS. However, you can download and install the libslax distribution on a computer with a UNIX-like operating system to develop SLAX scripts outside of Junos OS.

The following sections provide instructions for downloading and installing the libslax distribution. Review the **INSTALL** file that comes with the distribution for additional information.

1. Downloading the libslax Distribution on page 66
2. Installing the libslax Distribution on page 67

### Downloading the libslax Distribution

- To download the latest release of the libslax distribution source release tarball:

  1. Access the download list at the following URL:

     http://code.google.com/p/libslax/downloads/list

  2. Select the **libslax-*release*.tar.gz** file, and download the file to your computer.

- To download the source code for the libslax distribution:

  1. If your system does not a have a way to access a Subversion repository, install a plug-in or Subversion client application that is suitable for your operating system.

  2. Access the Google SVN repository at the following URL:

     http://code.google.com/p/libslax/source/checkout

  3. Download the source files to your computer.

## Installing the libslax Distribution

To install the libslax distribution:

1. Navigate to the directory where the libslax distribution is saved.
2. If you downloaded a prepared release, issue the following command to uncompress and extract the contents of the libslax distribution; otherwise, proceed to step 4.
   - On FreeBSD and Linux systems:

     **% tar zxf libslax-*release*.tar.gz**
   - On Solaris systems:

     **% gzip -dc libslax-*release*.tar.gz | tar xf**
   - In a Cygwin environment:

     **% tar -zxf libslax-*release*.tar.gz**
3. The previous command creates the **libslax-*release*** directory, so make this the working directory.

   **% cd libslax-*release***
4. Install any required packages for your specific system environment.
   - For Cygwin:

     Under category "Devel", install the packages make, gcc, bison, and libxml2-devel.

     Under category "GNOME", install the package libxslt-devel.

     Under category "Web" or "Net", install the package libcurl-devel.
   - For Ubuntu:

     Install libxml2-dev and libxslt-dev.
5. Run **./configure** and supply any desired options and associated values.

   For a listing of available options and default values, issue the **./configure --help** command.

   By default, libslax installs architecture-independent files, including extension library files, in the **/usr/local** directories. To specify an installation prefix other than **/usr/local** for all installation files, include the **--prefix=*prefix*** option and specify an alternate location. To install just the extension library files in a different, user-defined location, include the **--with-extensions-dir=*dir*** option and specify the location where the extension libraries will live.

   **% ./configure [OPTION]... [VAR=VALUE]...**
6. Make the images.

   **% make**
7. Test the images.

   **% make test**
8. Install the images.

If the **sudo** command is not supported in your environment, omit that part of the command.

```
% sudo make install
```

9. Verify the installation by viewing the SLAX processor help.

```
$ slaxproc --help
Usage: slaxproc [mode] [options] [script] [files]
    Modes:
        --check OR -c: check syntax and content for a SLAX script
        --format OR -F: format (pretty print) a SLAX script
        --run OR -r: run a SLAX script (the default mode)
        --slax-to-xslt OR -x: turn SLAX into XSLT
        --xslt-to-slax OR -s: turn XSLT into SLAX

    Options:
        --debug OR -d: enable the SLAX/XSLT debugger
        --empty OR -E: give an empty document for input
        --exslt OR -e: enable the EXSLT library
        --help OR -h: display this help message
        --html OR -H: Parse input data as HTML
        --include <dir> OR -I <dir>: search directory for includes/imports
        --indent OR -g: indent output ala output-method/indent
        --input <file> OR -i <file>: take input from the given file
        --lib <dir> OR -L <dir>: search directory for extension libraries
        --name <file> OR -n <file>: read the script from the given file
        --no-randomize: do not initialize the random number generator
        --output <file> OR -o <file>: make output into the given file
        --param <name> <value> OR -a <name> <value>: pass parameters
        --partial OR -p: allow partial SLAX input to --slax-to-xslt
        --trace <file> OR -t <file>: write trace data to a file
        --verbose OR -v: enable debugging output (slaxLog())
        --version OR -V: show version information (and exit)
        --write-version <version> OR -w <version>: write in version

Project libslax home page: http://libslax.googlecode.com
```

To support SLAX within an application, link with libslax and call the libslax initializer function:

```
#include <libslax/slax.h>

slaxEnable(1);
```

## libslax Library and Extension Libraries Overview



### libslax Library

libslax is an open-source implementation of the SLAX language using the "New BSD License". libslax is written in C and is built on top of the libxml2, libxslt, and libexslt libraries.

The core of the libslax distribution is the libslax library, which incorporates a SLAX parser to read SLAX files, a SLAX writer to write SLAX files, a debugger, a profiler, and the SLAX processor (slaxproc) command-line tool . The parser turns a SLAX source file into an

XSLT tree (xmlDocPtr) using the **xsltSetLoaderFunc()** hook. The writer turns an XSLT tree (xmlDocPtr) into a file containing SLAX statements.

## libslax Extension Libraries

libslax supports a way to dynamically load extension libraries. The libslax distribution includes the xutil, bit, and cURL extension libraries. The source files for the default extension libraries are stored in the **libslax-*release*/extensions** directory of the distribution. You can supply additional extension libraries beyond the default extension libraries supported by the libslax distribution. Extension library locations can be specified statically at build time or dynamically at run time.

By default, libslax installs architecture-independent files, including extension library files, in the **/usr/local** directories. Specifically, libslax installs the extension libraries in the **/usr/local/lib/slax/extensions** directory. If you do not specify a different installation directory for the extension libraries at build time, the SLAX processor checks this directory for extension libraries when executing a script.

There are several ways to specify extension library locations at build time. During installation, to specify a directory prefix other than **/usr/local** for all installation files, including the libraries, execute the **./configure** command and include the **--prefix=*prefix*** option specifying the location to install the files. The default extension libraries are installed in the ***prefix*/lib/slax/extensions** directory, and the SLAX processor checks this directory for extension libraries when executing a script. To install just the extension library files in a different, user-defined location, execute the **./configure** command and include the **--with-extensions-dir=*dir*** option specifying the location where the extension libraries live. The SLAX processor will then automatically check the specified directory for extension libraries when executing a script. For more information about installing libslax, see "Downloading and Installing the libslax Distribution" on page 66.

There are several ways to specify extension library locations dynamically after installation is complete. You can define or update the SLAXEXTPATH environment variable to include the directory locations of additional extension libraries. The variable value is a colon-separated list of directories. The SLAX processor automatically checks these directories for extension libraries when executing a script. Alternatively, you can specify the extension library location when you execute a script by using the **slaxproc** command with the **--lib** or **-L** option.

To summarize, extension library locations are supplied to the SLAX processor in one of the following ways:

- By default, in the **/usr/local/lib/slax/extensions** directory.
- In **lib/slax/extensions/** under the directory specified by the **./configure --prefix** option given at build time.
- In the user-defined directory specified by the **./configure --with-extension-dir** option given at build time.
- In a directory included in the colon-separated list of the SLAXEXTPATH environment variable.
- In a directory provided using the **--lib** or **-L** argument to the **slaxproc** command.

**Related Documentation**

- libslax bit Extension Library on page 70
- libslax cURL Extension Library on page 72
- Downloading and Installing the libslax Distribution on page 66
- libslax Distribution Overview on page 65

## libslax Default Extension Libraries: bit, cURL, and xutil



### libslax bit Extension Library

The bit extension library contains functions that create and manipulate bit strings. The functions support 64-bit integer arguments. To incorporate functions from the bit extension library into SLAX scripts, include the namespace statement for that library in the script.

```
ns bit extension = "http://xml.libslax.org/bit";
```

Call the bit extension functions using the **bit** prefix and the function name. For example:

```
version 1.1 ;
ns bit extension = "http://xml.libslax.org/bit";

var $a = 63;
var $b = { expr "10111"; }

match / {
  <out> {
    <bit-and> {
      <a1> bit:and("101100", "100101");
      <a2> bit:and($a, $b);
      <a3> bit:and($a, number($b));
    }
    <bit-or> {
      <a1> bit:or("101100", "100101");
      <a2> bit:or($a, $b);
      <a3> bit:or($a, number($b));
    }
    <bit-mask> {
      <a1> bit:mask(0);
      <a2> bit:mask(8, 32);
    }
    <ops> {
      <a1> bit:to-int("10101");
    }
  }
}
```

Table 7 on page 71 lists the functions available in the bit extension library and which are supported in SLAX 1.1 scripts.

Table 7: Functions in the bit Extension Library

| Function and Arguments | Description | Example |
|---|---|---|
| bit:and(b1, b2) | Return the logical AND of two bit strings. | bit:and("101100", "100101")<br>return value: "100100" |
| bit:clear(b1,bitnum) | Set the specified bit in the bit string to zero and return the new bit string. Bits are numbered starting from zero. If the integer argument is greater than the bit string length, the bit string is extended. | bit:clear("11111", 0)<br>return value: "11110"<br><br>bit:clear("11111", 6)<br>return value: "0011111" |
| bit:compare(value1, value2) | Compare two values and return an integer less than, equal to, or greater than zero, if the first argument is found to be less than, equal to, or greater than the second argument, respectively. | bit:compare("10000", 16)<br>return value: 0<br><br>bit:compare("11111", "10000")<br>return value: 1 |
| bit:from-hex(string, len?) | Return the value of the hex argument as a bit string. The optional second argument pads the bit string with leading 0's until it is the specified length. | bit:from-hex("0x45", 8)<br>return value: "01000101" |
| bit:from-int(integer, len?) | Return the value of the integer argument as a bit string. The optional second argument pads the bit string with leading 0's until it is the specified length. | bit:from-int(65,8)<br>return value: "01000001" |
| bit:mask(count, len?) | Return a bit string with count low-order bits set to one. The optional second argument pads the bit string with leading 0's until it is the specified length. | bit:mask(4, 8)<br>return value: "00001111" |
| bit:nand(b1, b2) | Return the logical NAND of two bit strings. | bit:nand("101100", "100101")<br>return value: "010010" |
| bit:nor(b1, b2) | Return the logical NOR of two bit strings. | bit:nor("101100", "100101")<br>return value: "011011" |
| bit:not(b1) | Return the inversion (NOT) of a bit string. | bit:not("101100")<br>return value: "010011" |
| bit:or(b1, b2) | Return the logical OR of two bit strings. | bit:or("101100", "100101")<br>return value: "101101" |

Table 7: Functions in the bit Extension Library *(continued)*

| Function and Arguments | Description | Example |
|---|---|---|
| bit:set(b1, bitnum) | Set the specified bit in the bit string and return the new bit string. Bits are numbered starting from zero. If the integer argument is greater than the bit string length, the bit string is extended. | bit:set("1001", 2)<br>return value: "1101"<br><br>bit:set("1001", 6)<br>return value: "1001001" |
| bit:to-int(b1) | Return the value of the bit string argument as an integer. | bit:to-int("101100")<br>return value: 44 |
| bit:to-hex(b1) | Return the value of the bit string argument as a string representation of the hex value. | bit:to-hex("101100")<br>return value: "0x2c" |
| bit:xor(b1, b2) | Return the logical XOR of two bit strings. | bit:xor("101100", "100101")<br>return value: "001001" |
| bit:xnor(b1, b2) | Return the logical XNOR of two bit strings. | bit:xnor("101100", "100101")<br>return value: "110110" |

## libslax cURL Extension Library



### Understanding the cURL Extension Library

cURL is a command-line tool that uses the libcurl library and permits data transfers using a number of protocols, including FTP, FTPS, HTTP, HTTPS, SCP, and SMTP. For more information about cURL, see the cURL website at http://curl.haxx.se/.

To incorporate functions from the cURL extension library into SLAX scripts, include the namespace statement for that library in the script.

```
ns curl extension = "http://xml.libslax.org/curl";
```

Call the cURL extension functions using the **curl** prefix and the function name. cURL operations are directed using a set of elements passed to the cURL extension functions.

Table 8 on page 73 lists the supported operations in the cURL extension library and includes a description of each function. Table 9 on page 73 and Table 10 on page 74 list the supported elements and include the syntax and a description of each element. Table

9 on page 73 lists elements used for web services operations, and Table 10 on page 74 lists the elements used for e-mail operations.

### Table 8: Functions in the cURL Extension Library

| Function | Description |
|---|---|
| curl:close | Close an open connection. Further operations cannot be performed over the connection. See "curl:close" on page 75. |
| curl:open | Open a connection to a remote server, allowing multiple operations over a single connection. See "curl:open" on page 76. |
| curl:perform | Perform simple transfers using a persistent connection handle provided by curl:open. See "curl:perform" on page 76. |
| curl:set | Record a set of parameters that persists for the lifespan of a connection. See "curl:set" on page 77. |
| curl:single | Perform transfer operations without using a persistent connection. See "curl:single" on page 77. |

### Table 9: Web Services Elements in the cURL Extension Library

| Element | Description | Syntax |
|---|---|---|
| <content-type> | Provide the MIME type for the transfer payload. | <content-type> "mime/type"; |
| <fail-on-error> | Indicate that the transfer should fail if any errors, including insignificant ones, are detected. | <fail-on-error>; |
| <format> | Specify the expected format of returned results, allowing the cURL extension to automatically make the content available in the native format. Formats include "html", "text", and "xml". | <format> "xml"; |
| <header> | Provide additional header fields for the request. | <header name="name"> "value"; |
| <insecure> | Indicate a willingness to tolerate insecure communications operations. Specifically, allow SSL Certs without checking the common name. | <insecure>; |

Table 9: Web Services Elements in the cURL Extension Library *(continued)*

| Element | Description | Syntax |
|---|---|---|
| <method> | Set the method used to transfer data. This controls the HTTP request type, as well as triggering other transfer mechanisms.<br><br>Acceptable method names include "get", "post", "delete", "head", "email", "put", and "upload". The "get" method is the default. | <method> "get"; |
| <param> | Provide additional parameter values for the request. These parameters are typically encoded into the URL. | <param name="x"> "y"; |
| <password> | Set the user's password for the transfer. | <password> "password"; |
| <secure> | Request the use of the secure version of a protocol, including HTTPS and FTPS. | <secure>; |
| <upload> | Indicate this is a file upload request. | <upload>; |
| <url> | Set the base URL for the request. | <url> "target-url"; |
| <username> | Set the username to use for the transfer. | <username> "username"; |
| <verbose> | Request detailed debug information about the operations and communication of the cURL transfer. | <verbose>; |

Table 10: E-mail Elements in the cURL Extension Library

| Element | Description | Syntax |
|---|---|---|
| <cc> | Set the "Cc" address for e-mail (SMTP) requests. For multiple addresses, use multiple <cc> elements. | <cc> "cc-user@email.example.com"; |
| <contents> | Specify the contents to be transferred. | <contents> "multi-\nline\ncontents\n"; |
| <from> | Set the "From" address for e-mail (SMTP) requests. | <from> "source-user@email.example.com"; |
| <header> | Provide additional header fields for the request. | <header name="name"> "value"; |
| <local> | Set the local hostname for e-mail (SMTP) requests. | <local> "local host name"; |

Table 10: E-mail Elements in the cURL Extension Library *(continued)*

| Element | Description | Syntax |
|---|---|---|
| <server> | Set the outgoing SMTP server name. Currently, MX records are not processed. | <server> "email-server.example.com"; |
| <subject> | Set the "Subject" field for e-mail (SMTP) requests. | <subject> "email subject string"; |
| <to> | Set the "To" address for e-mail (SMTP) requests. For multiple addresses, use multiple <to> elements. | <to> "to-user@email.examplecom"; |
| <verbose> | Request detailed debug information about the operations and communication of the cURL transfer. | <verbose>; |

The libcurl elements closely mimic the options used by the native C libcurl API in libcurl's **curl_easy_setopt()** function. Once the options are set, a call to **curl_easy_perform()** performs the requested transfer. For more information about the **curl_easy_setopt()** function, see http://curl.haxx.se/libcurl/c/curl_easy_setopt.html .

In the SLAX cURL extension library, the libcurl API options are represented as individual elements. For example, the **<url>** element is mapped to the CURLOPT_URL option , the **<method>** element is mapped to the CURLOPT_CUSTOMREQUEST option, and so forth.

These elements can be used in three ways:

- The **curl:single()** extension function permits a set of options to be used in a single transfer operation with no persistent connection handle.
- The **curl:perform()** extension function permits a set of options to be used with a persistent connection handle. The handle is returned from the **curl:open()** extension function and can be closed with the **curl:close()** extension function.
- The **curl:set()** extension function records a set of options for a connection handle and keeps those options active for the lifetime of the connection. For example, if the script needs to transfer a number of files, it can record the **<username>** and **<password>** options and avoid repeating them in every **curl:perform()** call.

### curl:close

The **curl:close()** extension function closes an open connection. Further operations cannot be performed over the connection once it is closed.

The syntax is:

```
node-set[empty] curl:close(node-set[connection]);
```

The argument is the connection handle to close.

### curl:open

The **curl:open()** extension function opens a connection to a remote server, allowing multiple operations over a single connection.

The syntax is:

node-set*[connection]* curl:open();

The returned object is a connection handle that can be passed to **curl:perform()** or **curl:close()**.

### curl:perform

The **curl:perform()** extension function performs simple transfers using a persistent connection handle provided by **curl:open()**.

The syntax is:

node-set*[object]* curl:perform(node-set*[connection]*, node-set*[options]*)

The arguments are the connection handle and a set of option elements. Supported cURL extension library elements are defined in Table 9 on page 73 and Table 10 on page 74.

The returned object is an XML hierarchy containing the results of the transfer. Table 11 on page 76 lists the possible elements in the reply, and Table 12 on page 76 lists the possible elements contained within the **<header>** element.

#### Table 11: curl:perform Reply Elements

| Element | Contents |
|---|---|
| <curl-success> | Empty element which Indicates success |
| <data> | Parsed data |
| <error> | Error message text, if any |
| <header> | Parsed header fields |
| <raw-data> | Raw data from the reply |
| <raw-headers> | Raw header fields from the reply |
| <url> | Requested URL |

#### Table 12: curl:perform <header> Elements

| Element | Contents |
|---|---|
| <code> | HTTP reply code |
| <field> | HTTP reply field (with @name and value) |

Table 12: curl:perform <header> Elements *(continued)*

| Element | Contents |
|---|---|
| <message> | HTTP reply message |
| <version> | HTTP reply version string |

The following example shows the **<header>** element with header fields parsed into **<field>** elements:

```
<header>
  <version>HTTP/1.1</version>
  <code>404</code>
  <message>Not Found</message>
  <field name="Content-Type">text/html</field>
  <field name="Content-Length">345</field>
  <field name="Date">Mon, 08 Aug 2011 03:40:21 GMT</field>
  <field name="Server">lighttpd/1.4.28 juisebox</field>
</header>
```

### curl:set

The **curl:set()** extension function records a set of parameters that persist for the lifespan of a connection.

The syntax is:

```
node-set[empty] curl:set(node-set[handle], node-set[options]);
```

The arguments are the connection handle and a set of option elements. Supported cURL extension library elements are defined in Table 9 on page 73 and Table 10 on page 74.

### curl:single

The **curl:single()** extension function performs transfer operations without using a persistent connection.

The syntax is:

```
node-set[result] curl:single(node-set[options]);
```

The returned object is identical in structure to the one returned by **curl:perform()**. Refer to "curl:perform" on page 76 for additional information.

### cURL Examples

The following examples show SLAX version 1.1 scripts that use the cURL extension library functions to perform operations.

The following SLAX script performs a simple GET operation to retrieve a web page. The script specifies the header field for the HTTP header and a parameter that is incorporated into the requested URL.

```
version 1.1;

ns curl extension = "http://xml.libslax.org/curl";
```

```
param $url = "http://www.juniper.net";

match / {
  <op-script-results> {
    var $options = {
      <header name="client"> "slaxproc";
      <param name="smokey"> "bandit";
    }
    var $results = curl:single($url, $options);
    message "completed: " _ $results/headers/message;
    <curl> {
      copy-of $results;
    }
  }
}
```

The following SLAX script takes a username and password and uses the Google login services to translate them into an "Authorization" string:

```
version 1.1;

ns curl extension = "http://xml.libslax.org/curl";

param $url = "https://www.google.com/accounts/ClientLogin";
param $username;
param $password;

var $auth-params := {
  <url> $url;
  <method> "post";
  <insecure>;
  <param name="Email"> $username;
  <param name="Passwd"> $password;
  <param name="accountType"> "GOOGLE";
  <param name="service"> "wise";
  <param name="source"> "test-app";
}

match / {
  var $curl = curl:open();
  var $auth-cred = curl:perform($curl, $auth-params);

  <options> {
    for-each(slax:break-lines( $auth-cred/raw-data )) {
      if(starts-with(.,"Auth")) {
        <header name="GData-Version"> "3.0";
        <header name="Authorization"> "GoogleLogin " _ .;
      }
    }
  }
```

```
  expr curl:close($curl);
}
```

The following SLAX script sends an e-mail by way of a server, which is provided as a parameter:

```
version 1.1;

ns curl extension = "http://xml.libslax.org/curl";

param $server;

match / {
  <out> {
    var $info = {
      <method> "email";
      <server> $server;
      <from> "muffin@example.com";
      <to> "phil@example.net";
      <subject> "Testing...";
      <contents> "Hello,
This is an email.
Thanks,
 Phil
";
    }
    var $res = curl:single($info);
    <res> {
      copy-of $res;
    }
  }
}
```

## libslax xutil Extension Library

The xutil extension library contains functions that convert between strings and XML node sets. To incorporate functions from the xutil extension library into SLAX scripts, include the namespace statement for that library in the script.

```
ns xutil extension = "http://xml.libslax.org/xutil";
```

Call the xutil extension functions using the **xutil** prefix and the function name. Table 13 on page 79 lists the functions available in the xutil extension library and which are supported in SLAX 1.1 scripts.

### Table 13: Functions in the xutil Extension Library

| Function and Arguments | Description |
|---|---|
| xutil:string-to-xml(string+) | Return a node set of the concatenated string arguments. |
| xutil:xml-to-string(node-set+) | Return a string representation of the XML hierarchies provided as arguments. |

**Related Documentation**

- libslax Library and Extension Libraries Overview on page 68

## Understanding the SLAX Processor (slaxproc)



### slaxproc Overview

The libslax distribution contains the libslax library, which incorporates a SLAX writer and SLAX parser, a debugger, a profiler, and the SLAX processor (slaxproc). The SLAX processor is a command-line tool that can validate SLAX script syntax, convert between SLAX and XSLT formats, and format, debug, or run SLAX scripts.

The SLAX processor is invoked on the command line using the **slaxproc** command. The **slaxproc** command accepts command-line arguments that specify the mode of the processor, any behavioral options, and required input and output files.

The syntax for the **slaxproc** command is:

```
slaxproc [mode] [options] [script] [files]
```

The slaxproc mode defines what function the processor performs. slaxproc options include file options and common options. File options are used to specify the script file, input file, output file, and trace file. Common options include additional functionality provided by the SLAX processor such as verbose debugging output.

You can access the slaxproc help by issuing the **slaxproc** command with the **--help** or **-h** option.

```
$ slaxproc -h
```

### slaxproc Modes

The slaxproc mode defines what function the processor performs. Valid modes include: **check**, which validates SLAX script syntax and content; **format**, which formats a script to correct the indentation and spacing to the preferred style; **run**, which executes a SLAX script; and **convert**, which converts a script between SLAX and XSLT formats.

Table 14 on page 81 outlines the slaxproc modes. The default mode is **--run** or **-r**. If you do not explicitly specify a mode, the SLAX processor executes a script.

Table 14: slaxproc Modes

| Mode | Description |
|---|---|
| --check<br>-c | Perform a syntax and content check on a SLAX script, reporting any errors. This mode is useful for off-box syntax checks before installing or uploading scripts to a device running Junos OS. |
| --format<br>-F | Format a SLAX script, correcting indentation and spacing to the preferred style. |
| --run<br>-r | Run a SLAX script. This is the default mode. The script name, input file name, and output file name can be provided using command-line options, positional arguments, or a mix of both. Input defaults to standard input, and output defaults to standard output. |
| --slax-to-xslt<br>-x | Convert a SLAX script into XSLT format. The script filename and output filename are provided using command-line options, positional arguments, or a mix of both. |
| --xslt-to-slax<br>-s | Convert an XSLT script into SLAX format. The script filename and output filename are provided using command-line options, positional arguments, or a mix of both. |

## slaxproc Options

The slaxproc options include file options and common options. File options are used specify the script file, input file, output file, and trace file. Common options include additional functionality and options provided by the SLAX processor such as verbose debugging output. Table 15 on page 81 lists the slaxproc common options and file options.

Table 15: slaxproc Common Options and File Options

| Option | Description |
|---|---|
| --debug<br>-d | Enable the SLAX/XSLT debugger. |
| --empty<br>-E | Provide an empty document as the input data set. This is useful for scripts that do not expect or need meaningful input. |
| --exslt<br>-e | Enable the EXSLT library, which provides a set of standard extension functions. See http://www.exslt.org for more information. |
| --help<br>-h | Display the help message and exit. |
| --html<br>-H | Parse input data using the HTML parser, which differs from XML. |
| --include <dir><br>-I <dir> | Add a directory to the list of directories searched when using include and import files. Alternatively, you can define the SLAXPATH environment variable to specify a colon-delimited list of directories to search. |

Table 15: slaxproc Common Options and File Options *(continued)*

| Option | Description |
|---|---|
| --indent<br>-g | Indent output. This option is identical to the behavior triggered by **output-method { indent 'true'; }**. |
| --input <file><br>-i <file> | Read input from the specified file. |
| --lib <dir><br>-L <dir> | Add a directory to the list of directories searched when using extension libraries. Alternatively, you can define the SLAXEXTPATH environment variable to specify a colon-delimited list of extension library locations to search. |
| --name <file><br>-n <file> | Read the SLAX script from the specified file. |
| --no-randomize | Do not initialize the random number generator. This is useful if you want the script to return identical data for a series of invocations. This option is typically only used during testing. |
| --no-tty | Do not use tty for the SLAX debugger and other tty-related input needs. |
| --output <file><br>-o <file> | Write output to the specified file. |
| --param <name> <value><br>-a <name> <value> | Pass a parameter to the script using the name and value pair provided. Note that all parameters are string parameters, so normal quoting rules apply. |
| --partial<br>-p | Allow the input data to contain a partial SLAX script, which can be used with the **--slax-to-xslt** or **-x** mode to perform partial transformations. |
| --trace <file><br>-t <file> | Write trace data to the specified file. |
| --verbose<br>-v | Add verbose internal debugging output to the trace data output, including calls to the **slaxLog()** function. |
| --version<br>-V | Show version information and exit. |
| --write-version <version><br>-w <version> | Write the specified version number to the output file when converting a script using the **--xslt-to-slax** or **-s** mode. This can be used to limit the conversion to avoid using SLAX 1.1 features. Acceptable values are 1.0 and 1.1. If this option is not specified, the SLAX script version defaults to 1.1. |

## slaxproc File Argument Handling

For all modes except **check**, you have the option to reference file arguments positionally or use the file options to specify input and output files. If you use the file options, the files can be referenced in any order on the command line, and the file options can be interspersed among other command-line options.

If no input file is required, use the **-E** option to indicate an empty input document. Additionally, if the input or output option argument has the value "-", the standard input or standard output file is used. When using standard input, press Ctrl+d to signal the end-of-file.

To reference files positionally on the command line, specify the script file first if it is required for that mode, then specify the input file, and lastly specify the output file. Referencing the files positionally allows slaxproc to be plug compatible with xsltproc.

```
$ slaxproc script.slax input.xml output.xml
```

To reference files using explicit file option values, include **--name** or **-n**, **--input** or **-i**, and **--output** or **-o**, to specify the SLAX script file, and the input and output files, respectively.

```
$ slaxproc -i input.xml -n script.slax -o output.xml
```

If a file option is not provided, the filename is parsed positionally. In the following command, the input and output filenames are specified using the file options, but the script filename is referenced positionally:

```
$ slaxproc -i input.xml -o output.xml -g -v script.slax
```

To execute a script that requires no input file, include the **-E** option to indicate an empty input document.

```
$ slaxproc -E script.slax output.xml
```

## slaxproc UNIX Scripting Support

SLAX supports the "#!" UNIX scripting mechanism, allowing the first line of a script to begin with the characters '#' and '!' followed by a path to the executable that runs the script and a set of command-line arguments. For example:

```
#!/usr/bin/slaxproc -n
```

or

```
#!/opt/local/bin/slaxproc -n
```

The operating system adds the name of the scripts and any command-line arguments to the command line that follows the "#!". Adding the **-n** option allows additional arguments to be passed in on the command line. Flexible argument parsing allows aliases. For example, if the first line of the script is:

```
#!/usr/bin/slaxproc -E -n
```

additional arguments can be provided:

```
$ that-script -g output.xml
```

and the resulting command becomes:

```
/usr/bin/slaxproc -E -n /path/to/that-script -g output.xml
```

If the input or output argument has the value "-", the standard input or standard output file is used. This allows slaxproc to be used as a traditional UNIX filter.

**Related Documentation**

- Using the SLAX Processor (slaxproc) on page 84
- libslax Distribution Overview on page 65
- libslax Library and Extension Libraries Overview on page 68
-

## Using the SLAX Processor (slaxproc)

The SLAX processor (slaxproc) is a command-line tool that can validate SLAX script syntax, convert between SLAX and XSLT formats, and format or run SLAX scripts. The slaxproc mode options define what function the processor performs. The following sections outline each of the modes:

- Validating SLAX Script Syntax on page 84
- Converting Scripts Between XSLT and SLAX Formats on page 84
- Running SLAX Scripts on page 86
- Formatting SLAX Scripts on page 87

### Validating SLAX Script Syntax

The SLAX processor provides an option to check the syntax of a SLAX script.

- To check the syntax of a SLAX script, issue the **slaxproc** command with the **--check** or **-c** mode option and the script filename.

```
$ slaxproc --check script1.slax

 OR

$ slaxproc -c script1.slax
```

If the script syntax is correct, the SLAX processor issues a "script check succeeds" message. Otherwise, the processor issues a list of error messages detected during script parsing. Fix any indicated errors, and repeat the check.

### Converting Scripts Between XSLT and SLAX Formats

The SLAX processor supports converting scripts between SLAX and XSLT formats. When you convert a script, you have the option to reference the file arguments positionally or use the command-line file options, **--input** or **-i** and **--output** or **-o**, to specify the original input script and the converted output script, respectively. If you use the command-line file options, the files can be referenced in any order on the command line, and the file options can be interspersed among other command-line options.

If you do not provide an argument specifying an input file or an output file, the standard input or standard output file is used. When using standard input, press Ctrl+d to signal the end-of-file.

To convert a SLAX script to XSLT, issue the **slaxproc** command with the **--slax-to-xslt** or **-x** mode option. To reference the files positionally, specify the input SLAX file as the first argument and the desired output path and filename of the converted XSLT script as the second argument. To reference the files using command-line file options, include the file options in any order. For example:

```
$ slaxproc --slax-to-xslt  test/script2.slax  test/script2.xsl

 OR

$ slaxproc -x  -i test/script2.slax  -o test/script2.xsl
```

To convert an XSLT script to SLAX, issue the **slaxproc** command with the **--xslt-to-slax** or **-s** mode option. To reference the files positionally, specify the input XSLT file as the first argument and the desired output path and filename of the converted SLAX script as the second argument. To reference the files using command-line file options, include the file options in any order.

Optionally, when converting a script from XSLT to SLAX, include the **--write-version** or **-w** option to specify the SLAX version of the converted script. Acceptable values are 1.0 and 1.1. The default is version 1.1. Use the **-p** option for partial input when you do not require the SLAX script boilerplate in the output.

The following example converts the XSLT script **script1.xsl** to the SLAX script **script1.slax**. The SLAX script will include the statement "version 1.0;" as the first line of the script.

```
$ slaxproc --xslt-to-slax -w 1.0 test/script1.xsl  test/script1.slax

 OR

$ slaxproc -s -w 1.0 -i test/script1.xsl -o test/script1.slax
```

The slaxproc **--xslt-to-slax** mode with the **-p** option is useful for quickly converting Junos OS hierarchies from XML format into SLAX. The following example provides the Junos OS **[edit policy-options]** hierarchy in XML format as input to the SLAX processor. The **-p** option indicates partial script input as opposed to a full script.

```
$ slaxproc -s -p
<policy-options>
  <policy-statement>
    <name>export-policy</name>
    <term>
      <name>term1</name>
      <from>
        <route-filter>
          <address>10.0.4.4/30</address>
          <prefix-length-range>/30-/30</prefix-length-range>
        </route-filter>
      </from>
      <then>
        <accept/>
      </then>
    </term>
```

```
  </policy-statement>
</policy-options>
[Ctrl+d]
```

The SLAX processor returns the SLAX formatting for the hierarchy.

```
<policy-options> {
  <policy-statement> {
    <name> "export-policy";
    <term> {
      <name> "term1";
      <from> {
        <route-filter> {
          <address> "10.0.4.4/30";
          <prefix-length-range> "/30-/30";
        }
      }
      <then> {
        <accept>;
      }
    }
  }
}
```

## Running SLAX Scripts

The SLAX processor supports executing SLAX scripts from the command line. This is the default slaxproc mode. To explicitly use this mode, issue the **slaxproc** command with the **--run** or **-r** command-line mode option.

When you execute a script, you have the option to reference the file arguments positionally or use the command-line file options, **--name** or **-n**, **--input** or **-i**, and **--output** or **-o**, to specify the SLAX script file, and the input and output files, respectively. If you use the command-line file options, the files can be referenced in any order on the command line, and the file options can be interspersed among other command-line options.

If no input file is required, use the **-E** option to indicate an empty input document. Additionally, if the input or output argument has the value "-", the standard input or standard output file is used. When using standard input, press Ctrl+d to signal the end-of-file.

The syntax for executing a script is:

```
$ slaxproc script input-file output-file
```

or

```
$ slaxproc (--name | -n) script (--input | -i) input-file (--output | -o) output-file
```

To execute a script using the slaxproc command-line tool:

1. Create a script using your favorite editor.
2. (Optional) Check the script syntax by invoking the processor with the **--check** or **-c** mode option, and fix any indicated errors.

   ```
   $ slaxproc -c test/script1.slax
   ```

3. Execute the script and provide the required input and output files as well as any desired slaxproc options.

   You can reference files positionally or use the command-line file options.

   - To execute a script named **script1.slax** using **input.xml** as the input document and **output.xml** as the output document, issue either of the following commands. The two commands are identical in execution.

     ```
     $ slaxproc script1.slax input.xml output.xml

     $ slaxproc -n script1.slax -i input.xml -o output.xml
     ```

   - To execute a script that requires no input file, include the **-E** option to indicate an empty input document. For example:

     ```
     $ slaxproc -E script1.slax output.xml

     $ slaxproc -n script1.slax -o output.xml -E
     ```

   - To execute a script and use standard input as the input document, issue the **slaxproc** command with no input file argument. At the prompt, enter the input and press Ctrl+d to signal the end-of-file. For example:

     ```
     $ slaxproc -n script1.slax -o output.xml
     <user input>
     [Ctrl+d]
     ```

## Formatting SLAX Scripts

The SLAX processor provides the option to format a script to correct the indentation and spacing to the preferred style. When you format a script, you have the option to reference the file arguments positionally or use the command-line file options, **--input** or **-i** and **--output** or **-o**, to specify the unformatted input file and the formatted output file, respectively. If you use the command-line file options, the files can be referenced in any order on the command line.

To format a SLAX script, issue the **slaxproc** command with the **--format** or **-F** mode option. To reference the files positionally, specify the unformatted SLAX script as the first argument and the desired output path and filename of the formatted SLAX script as the second argument. To reference the files using command-line file options, include the file options in any order. For example:

```
$ slaxproc --format  script1.slax  script1-format.slax

 OR

$ slaxproc -F  -i script1.slax  -o script1-format.slax
```

Given the following unformatted SLAX script as input:

```
version 1.1;

decimal-format default-format {
decimal-separator "." ;
digit "#" ;
grouping-separator "," ;
```

```
infinity "Infinity" ;
minus-sign "-" ;
nan "NaN";
pattern-separator ";" ;
percent "%";
per-mille "\x2030";
zero-digit "0" ;
}

match / {
var $number = -14560302.5;
expr format-number($number, "###,###.00", "default-format");
}
```

the SLAX processor outputs the following formatted SLAX script:

```
version 1.1;

decimal-format default-format {
  decimal-separator ".";
  digit "#";
  grouping-separator ",";
  infinity "Infinity";
  minus-sign "-";
  pattern-separator ";";
  percent "%";
  per-mille " 30";
  zero-digit "0";
  nan "NaN";
}

match / {
  var $number = -14560302.5;

  expr format-number($number, "###,###.00", "default-format");
}
```

**Related Documentation**

- libslax Distribution Overview on page 65
- libslax Library and Extension Libraries Overview on page 68
- Understanding the SLAX Processor (slaxproc) on page 80

## SLAX Debugger, Profiler, and callflow

- SLAX Debugger, Profiler, and callflow Overview on page 88
- Using the SLAX Debugger, Profiler, and callflow on page 90

### SLAX Debugger, Profiler, and callflow Overview

The Junos OS command-line interface (CLI) and the libslax distribution include the SLAX debugger (sdb), which is used to trace the execution of SLAX scripts. The SLAX debugger enables you to step through script execution, pause script execution at defined breakpoints, and review the value of script variables at any point.

The SLAX debugger operation and command syntax resemble that of the GNU Project Debugger (GDB). Many of the sdb commands follow their GDB counterparts, to the extent possible.Table 16 on page 89 lists the SLAX debugger commands and a brief description of each command.

The SLAX debugger includes a profiler that can report information about the activity and performance of a script. The profiler, which is automatically enabled when you start the debugger, tracks script execution until the script terminates. At any point, profiling information can be displayed or cleared, and the profiler can be temporarily disabled or enabled. The SLAX debugger **callflow** command enables printing of informational data when you enter or exit levels of the script.

### Table 16: SLAX Debugger Commands

| Command | Description |
|---|---|
| break [*loc*] | Add a breakpoint to the script at the current line of execution. Optionally specify [*file*:]*line* or a template name to create a breakpoint at that position. |
| callflow [on \| off] | Enable or disable callflow tracing. You can explicitly specify the **on** or **off** value. Omitting the value toggles callflow on and off. |
| continue [*loc*] | Continue running the script until it reaches the next breakpoint. If there are no defined breakpoints, the script runs in its entirety. Optionally, specify [*file*:]*line* or a template name. When you include the optional argument, script execution continues until it reaches either a breakpoint or the specified line number or template name, whichever comes first. |
| delete [*num*] | Delete one or all breakpoints. Breakpoints are numbered sequentially as they are created. Omit the optional argument to delete all breakpoints. Include the breakpoint number as an argument to delete only the specified breakpoint. View currently active breakpoints with the **info** command. |
| finish | Finish executing the current template. |
| help | Display the help message. |
| info [breakpoints \| profile \| profile brief] | Display information about the current script. The default command lists all breakpoints in the script. Optionally specify the **profile** or **profile brief** arguments to display profiling information. |
| list [*loc*] | List the contents of the current script. Optionally specify [*file*:]*line* or a template name from which point the debugger lists partial script contents. The output includes the filename, line number, and code. |
| next | Execute the next instruction, stepping over any function or template calls. |
| over | Execute the next instruction, stepping over any function or template calls or instruction hierarchies. |
| print <*xpath*> | Print the value of the XPath expression. |

Table 16: SLAX Debugger Commands *(continued)*

| Command | Description |
|---|---|
| profile [clear \| on \| off \| report \| report brief] | Enable or disable the profiler. The profiler is enabled by default. Include the **clear** option to clear profiling information. Include the **report** or **report brief** option to display profiling information for the current script. |
| quit | Exit debugging mode. |
| reload | Reload the script. |
| run | Restart script execution from the beginning of the script. |
| step | Execute the next instruction, stepping into any function or template calls or instruction hierarchies. |
| where | Show the backtrace of template calls. |

## Using the SLAX Debugger, Profiler, and callflow



### Invoking the SLAX Debugger

Both the Junos OS CLI and the SLAX processor in the libslax distribution include the SLAX debugger (sdb), which is used to trace the execution of SLAX scripts.

When you invoke the SLAX debugger, the command-line prompt changes to (sdb) to indicate that you are in debugging mode. For example:

```
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

When using the SLAX debugger from the Junos OS CLI, you can only use the debugger with op scripts that are enabled in the configuration. To invoke the SLAX debugger from the CLI on a device running Junos OS, issue the **op invoke-debugger cli** operational mode command, include the op script name, and optionally include any necessary script arguments.

user@host>**op invoke-debugger cli** ***script*** **<*argument-name argument-value*>**

The following example invokes the SLAX debugger for the op script **ge-interfaces.slax**, which has two parameters, **interface** and **protocol**. Values are supplied for both arguments.

```
user@host> op invoke-debugger cli ge-interfaces interface ge-0/2/0.0 protocol inet
```

```
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

To invoke the SLAX debugger when using the SLAX processor, issue the **slaxproc** command with the **--debug** or **-d** option. Specify the script file and any input or output files. If no input file is required, use the **-E** option to indicate an empty input document. If the **-i** or **--input** argument has the value "-", or if you do not include the input option or an input file, standard input is used. When using standard input, press Ctrl+d to signal the end-of-file. The general syntax is:

```
$ slaxproc --debug [options] [script] [files]
```

The following example invokes the SLAX debugger for the script **script1.slax** with an empty input document and an output file **script1-output.xml**

```
$ slaxproc --debug -n script1.slax -o script1-output.xml -E
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

## Using the SLAX Debugger (sdb)

To view the SLAX debugger help message, issue the **help** command at the (sdb) prompt. To display the help message for a single command, issue **help** ***command***, where *command* is the sdb command for which you want more information. For example:

```
(sdb) help break
 break [loc]    Add a breakpoint at [file:]line or template
```

The process for debugging a script varies depending on the script. A generic outline is presented here:

1. Enter debugging mode.
2. Insert breakpoints in the script using the **break** command.

   During execution, the debugger pauses at defined breakpoints.

   The breakpoint location can be the name of a template or a line number in the current script, or the filename and a line number separated by a colon. If you do not include an argument, a breakpoint is created at the current line of execution. Breakpoints are numbered sequentially as you create them. To view a list of breakpoints, issue the **info breakpoints** command. To delete a breakpoint, issue the **delete** ***num*** command, and specify the breakpoint number. To delete all breakpoints, issue the **delete** command with no argument.

   The following example creates three breakpoints, the first at line 7, the second at line 25, and the third at the template named "three":

   ```
   (sdb) break 7
   Breakpoint 1 at file script1.slax, line 7
   (sdb) break 25
   Breakpoint 2 at file script1.slax, line 25
   (sdb) break three
   Breakpoint 3 at file script1.slax, line 51
   (sdb) info breakpoints
   ```

```
List of breakpoints:
  #1 [global] at script1.slax:7
  #2 template two at script1.slax:25
  #3 template three at script1.slax:51
```

3. Increment script execution by issuing the **continue**, **finish**, **next**, **over**, and **step** commands at the debugger prompt.

   For example:

   ```
   (sdb) next
   Reached breakpoint 1, at script1.slax:7
   script1.slax:3: var $byte = "10011001";
   ```

4. Review the value of variables as the program executes to ensure that they have the expected value.

   print *xpath-expression*

5. To reload the script contents at any point and restart script execution from the beginning, issue the **reload** command.

   ```
   (sdb) reload
   The script being debugged has been started already.
   Reload and restart it from the beginning? (y or n) y
   Reloading script...
   Reloading complete.
   ```

## Using the SLAX Profiler

The SLAX debugger includes a profiler that can report information about the activity and performance of a script. The profiler, which is automatically enabled when you start the debugger, tracks script execution until the script terminates. At any point, profiling information can be displayed or cleared, and the profiler can be temporarily disabled or enabled.

To access the profiler, issue the **profile** command at the SLAX debugger prompt, (sdb), and include any options. The profile command syntax is:

```
(sdb) profile [options]
```

Table 17 on page 92 lists the profile command options. Issuing the **profile** command with no additional options toggles the profiler on and off.

```
(sdb) profile
Disabling profiler
(sdb)
```

You can access the profiler help by issuing the **help profile** command at the (sdb) prompt.

### Table 17: Profile command options

| Option | Description |
|---|---|
| clear | Clear profiling information |
| off | Disable profiling |

Table 17: Profile command options *(continued)*

| Option | Description |
|---|---|
| on | Enable profiling |
| report [brief] | Report profiling information |

To enable the profiler and print a report:

1. Enter debugging mode. The profiler is enabled by default.
2. Step through script execution, or execute a script in its entirety.

```
(sdb) run
<?xml version="1.0"?>
<message>Down rev PIC in Fruvenator, Fru-Master 3000</message>
Script exited normally.
```

3. At any point during script execution, display profiling information.

   The **brief** option instructs sdb to avoid showing lines that were not hit, since there is no valid information. If you omit the **brief** option, dashes are displayed.

   (sdb) profile report brief

   The following sample output shows a profile report with and without the **brief** option. The source code data in the example is truncated for display purposes.

```
(sdb) profile report
 Line   Hits   User    U/Hit  System    S/Hit Source
    1      -      -        -       -        - version 1.0;
    2      -      -        -       -        -
    3      2      4     2.00       8     4.00 match / {
    4      1     25    25.00      13    13.00     var ....
    5      -      -        -       -        -
    6      -      -        -       -        -     for-each....
    7      1     45    45.00      10    10.00          ..
    8      1     12    12.00       5     5.00         <message>
    9      1     45    45.00      15    15.00          ....
   10      -      -        -       -        -     }
   11      -      -        -       -        - }
Total      6    131               51   Total

(sdb) pro rep b
 Line   Hits   User    U/Hit  System    S/Hit Source
    3      2      4     2.00       8     4.00 match / {
    4      1     25    25.00      13    13.00     var  ....
    7      1     45    45.00      10    10.00          ....
    8      1     12    12.00       5     5.00      <message>
    9      1     45    45.00      15    15.00          ....
Total      6    131               51   Total
```

The profile report includes the following information:

- **Line**—Line number in the source file.
- **Hits**—Number of times this line was executed.

- **User**—Number of microseconds of "user" time spent processing this line.
- **U/Hit**—Average number of microseconds of "user" time per hit.
- **System**—Number of microseconds of "system" time spent processing this line.
- **S/Hit**—Average number of microseconds of "system" time per hit.
- **Source**—Source code line.

This information not only shows how much time is spent during code execution, but can also show which lines are being executed, which can help debug scripts where the execution does not match expectations.

### Using callflow

The SLAX debugger **callflow** command enables printing of informational data when you enter or exit levels of the script.

To enable callflow and view callflow data for a script:

1. Enter debugging mode.
2. Issue the **callflow** command at the SLAX debugger prompt, (sdb).

   ```
   (sdb) callflow
   Enabling callflow
   ```

3. Step through script execution, or execute a script in its entirety.

   Callflow prints information as it enters and exits different levels of the script. Each output line references the instruction, filename, and line number of the frame.

   ```
   (sdb) run
   callflow: 0: enter <xsl:template> in match / at script3.slax:5
   callflow: 1: enter <xsl:template> in template one at script3.slax:14
   callflow: 2: enter <xsl:template> in template two at script3.slax:20
   callflow: 3: enter <xsl:call-template> at script3.slax:22
   ...
   <?xml version="1.0"?>
   <message>Down rev PIC in Fruvenator, Fru-Master 3000</message>
   Script exited normally.
   ```

**Related Documentation**

- op invoke-debugger cli
- Understanding the SLAX Processor (slaxproc) on page 80
- Using the SLAX Processor (slaxproc) on page 84
- libslax Distribution Overview on page 65

CHAPTER 7

# Junos Script Automation: Extension Functions, Templates, and Parameters

Junos OS automation features and the libslax library include extension functions and predefined templates and parameters that can be used in all scripts. The library includes extension functions that accomplish scripting tasks more easily, and an import file, **junos.xsl**, which includes named templates and predefined parameters that make scripts easier to read and write. This chapter discusses the extension functions, templates, and parameters in detail in the following sections:



## Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces

The Junos OS and SLAX extension functions are used in commit, op, and event scripts to accomplish scripting tasks more easily. Extension functions allow you to perform operations that are difficult or impossible to perform in XPath. The libraries provide logic, data manipulation, input and output, and utility functions.

Junos OS extension functions have functionality that is specific to devices running Junos OS. Junos OS extension functions are defined in the namespace with the associated Uniform Resource Identifier (URI) http://xml.juniper.net/junos/commit-scripts/1.0 . To use Junos OS extension functions in scripts, you must include the namespace URI in your style sheet declaration. Generally, the **jcs** prefix is mapped to the URI, and you then use the extension functions by prepending the **jcs** prefix to the function name. This avoids

name conflicts with standard XSLT functions. During processing, the **jcs** prefix is expanded into the URI reference.

The following SLAX namespace statement maps the **jcs** prefix to the namespace URI that defines Junos OS extension functions used in commit, op, and event scripts:

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

SLAX extension functions are defined in the namespace with the associated URI http://xml.libslax.org/slax . To use SLAX extension functions in scripts, you must include the namespace URI in your style sheet declaration. Generally, the **slax** prefix is mapped to the URI, and you then use the extension functions by prepending the **slax** prefix to the function name. During processing, the **slax** prefix is expanded into its associated URI reference.

The following SLAX namespace statement maps the **slax** prefix to the namespace URI that defines SLAX extension functions:

```
ns slax = "http://xml.libslax.org/slax";
```

The **slax** namespace is supported starting with Junos OS Release 12.2. Scripts using Junos OS-independent extension functions that existed in earlier releases in the **jcs** namespace can use either the **jcs** or the **slax** namespace starting with Junos OS Release 12.2. However, to use any of these functions in earlier Junos OS releases, scripts must use the **jcs** namespace URI. For a list of available functions and associated namespaces, see "Summary of Extension Functions in the jcs and slax Namespaces" on page 97.

To call an extension function in a script, include any required variable declarations, call the function using **jcs:*function-name*()** or **slax:*function-name*()** as appropriate, and pass along any required or optional arguments. Arguments must be passed into the function in the precise order specified by the function definition. This is different from a template, where the parameters are assigned by name and can appear in any order. The return value of an extension function must always either be assigned to a variable or designated as output.

The following example maps the **jcs** prefix to the namespace identified by the URI http://xml.juniper.net/junos/commit-scripts/1.0 . The script then calls the **jcs:invoke()** function with one argument.

**XSLT Syntax**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
    ...
    <xsl:variable name="result" select="jcs:invoke($command)"/>
    ...
</xsl: stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
    ...
    var $result = jcs:invoke($command);
    ...
```

The following example maps the **slax** prefix to the namespace identified by the URI http://xml.libslax.org/slax . The script then calls the **slax:get-input()** function with one string argument. The version statement specifies version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**SLAX Syntax**

```
version 1.1;
ns slax = "http://xml.libslax.org/slax";
    ...
    var $input = slax:get-input($prompt);
    ...
```

**Related Documentation**

- Summary of Extension Functions in the jcs and slax Namespaces on page 97
- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137
- SLAX Variables Overview on page 53
- XSLT Variables Overview on page 29

## Summary of Extension Functions in the jcs and slax Namespaces

The Junos OS and SLAX extension functions are summarized in Table 18 on page 97.

A function in the **jcs** namespace is defined in the namespace URI http://xml.juniper.net/junos/commit-scripts/1.0 .

A function in the **slax** namespace is defined in the namespace URI http://xml.libslax.org/slax . Functions using the **slax** namespace are supported starting with Junos OS Release 12.2.

Functions introduced in version 1.0 of the SLAX language can be used in SLAX scripts that include either a "version 1.0" statement or a "version 1.1" statement, but functions introduced in version 1.1 of the SLAX language can only be used in SLAX scripts that include the "version 1.1" statement. Scripts written in version 1.1 of the SLAX language are supported starting in Junos OS Release 12.2.

### Table 18: Extension Functions in the jcs and slax Namespaces

| Function | Name-spaces | SLAX Version | Type | Description |
|---|---|---|---|---|
| base64-decode() | slax | 1.1 | Data manipulation | Decode BASE64 encoded data and return a string. |
| base64-encode() | slax | 1.1 | Data manipulation | Encode a string of data in the BASE64 encoding format. |
| break-lines() | jcs, slax | 1.0 | Data manipulation | Break a simple element into multiple elements, delimited by newlines. |

Table 18: Extension Functions in the jcs and slax Namespaces *(continued)*

| Function | Name-spaces | SLAX Version | Type | Description |
|---|---|---|---|---|
| close() | jcs | 1.0 | Utility | Close a previously opened connection handle. |
| dampen() | jcs, slax | 1.0 | Utility | Prevent the same operation from being repeatedly executed within a script. |
| document() | slax | 1.1 | Input/output control | Read data from a file or URL and return a string. |
| empty() | jcs, slax | 1.0 | Logic | Evaluate a node set or string argument to determine if it is an empty value. |
| evaluate() | slax | 1.1 | Input/output control | Evaluate a SLAX expression and return the result. |
| execute() | jcs | 1.0 | Utility | Execute a remote procedure call (RPC) within the context of a specified connection handle. |
| first-of() | jcs, slax | 1.0 | Logic | Return the first nonempty (non-null) item in a list, If all objects in the list are empty, the default expression is returned. |
| get-command() | jcs, slax | 1.1 | Input/output control | Prompt the user for command input and return the input as a string. |
| get-hello() | jcs | 1.0 | Utility | Return the session ID and the capabilities of the NETCONF server during a NETCONF session. |
| get-input() | jcs, slax | 1.0 | Input/output control | Invoke a CLI prompt and wait for user input. If the script is run non-interactively, the function returns an empty value. This function cannot be used with event scripts. |
| get-protocol() | jcs | 1.0 | Utility | Return the session protocol associated with the connection handle. |
| get-secret() | jcs, slax | 1.0 | Input/output control | Invoke a CLI prompt and wait for user input. The input is not echoed back to the user. |

Table 18: Extension Functions in the jcs and slax Namespaces *(continued)*

| Function | Name-spaces | SLAX Version | Type | Description |
|---|---|---|---|---|
| hostname() | jcs | 1.0 | Utility | Return the fully qualified domain name associated with a given IPv4 or IPv6 address, provided the DNS server is configured on the device. |
| invoke() | jcs | 1.0 | Utility | Invoke an RPC on a local device running Junos OS. |
| open() | jcs | 1.0 | Utility | Return a connection handle that can be used to execute RPCs. |
| output() | jcs, slax | 1.0 | Input/output control | Generate unformatted output text that is immediately sent to the CLI session. |
| parse-ip() | jcs | 1.0 | Data manipulation | Parse an IPv4 or IPv6 address and return the host IP address, protocol family, prefix length, network address, and network mask. |
| printf() | jcs, slax | 1.0 | Input/output control | Generate formatted output text. Most standard **printf** formats are supported, in addition to some Junos OS-specific formats. The function returns a formatted string but does not print it on call. |
| progress() | jcs, slax | 1.0 | Input/output control | Issue a progress message containing the single argument immediately to the CLI session provided that the **detail** flag was specified when the script was invoked. |
| regex() | jcs, slax | 1.0 | Data manipulation | Evaluate a regular expression against a given string argument and return any matches. |
| sleep() | jcs, slax | 1.0 | Utility | Cause the script to sleep for a specified time. |
| split() | jcs, slax | 1.0 | Data manipulation | Split a string into an array of substrings delimited by a regular expression pattern. |
| sysctl() | jcs, slax | 1.0 | Utility | Return the value of the given **sysctl** value as a string or an integer. |
| syslog() | jcs, slax | 1.0 | Input/output control | Log messages with the specified priority to the system log file. |

Table 18: Extension Functions in the jcs and slax Namespaces *(continued)*

| Function | Name-spaces | SLAX Version | Type | Description |
|---|---|---|---|---|
| trace() | jcs, slax | 1.0 | Input/output control | Issue a trace message, which is sent to the trace file. |

**Related Documentation**

- Junos Named Templates in the jcs Namespace Summary on page 125
- Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces on page 95
- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137

## Extension Functions in the jcs and slax Namespaces

The Junos extension functions are discussed in detail in the following sections:

## base64-decode( ) Function (slax Namespace)

**Namespaces** http://xml.libslax.org/slax

**SLAX Syntax** *string* slax:base64-encode(*string*, <*control-string*>)

**Release Information** Function introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Decode BASE64 encoded data. BASE64 is a means of encoding arbitrary data into a radix-64 format that is more easily transmitted, typically using STMP or HTTP.

Include the optional control string argument to replace any non-XML control characters in the decoded string with the specified string. If the argument is an empty string, non-XML characters are removed. The decoded data is returned to the caller.

**Parameters** *control-string*—(Optional) String to replace non-XML control characters in the decoded string. Use an empty string argument to remove the non-XML characters.

*string*—BASE64 encoded data.

**Return Value** *string*—Decoded data.

—

**Usage Examples**

```
var $real-data = slax:base64-decode($encoded-data, "@");
```

## base64-encode( ) Function (slax Namespace)

**Namespaces** http://xml.libslax.org/slax

**SLAX Syntax** *string* slax:base64-encode(*string*)

**Release Information** Function introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Encode a string of data in the BASE64 encoding format. BASE64 is a means of encoding arbitrary data into a radix-64 format that is more easily transmitted, typically using STMP or HTTP.

**Parameters** *string*—Input data string.

**Return Value** *string*—Encoded data.

**Usage Examples**

```
var $encoded-data = slax:base64-encode($real-data);
```

## break-lines( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $lines = *prefix*:break-lines(*expression*);

**XSLT Syntax** <xsl:variable name="lines" select="*prefix*:break-lines(*expression*)"/>

**Release Information** Function introduced in Junos OS Release 7.6
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Break a simple element into multiple elements, delimited by newlines. This is especially useful for large output elements such as those returned by the **show pfe** command.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *expression*—Original output.

**Return Value** $lines—Output broken up into lines.

**Usage Examples**

```
var $lines = jcs:break-lines($output);
for-each ($lines) {
  ...
}
```

## close( ) Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $results = jcs:close(*connection*);

**XSLT Syntax** <xsl:variable name="results" select="jcs:close(*connection*)"/>

**Release Information** Function introduced in Junos OS Release 9.3.

**Description** Close a previously opened connection handle.

**Parameters** *connection*—Connection handle generated by a call to the **jcs:open( )** function.

**Usage Examples** The following example closes the connection handle **$connection**, which was originally generated by a call to the **jcs:open( )** function:

```
var $connection = jcs:open();
...
var $result = jcs:close($connection);
```

## dampen() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $result = *prefix*:dampen(*tag-string*, *max*, *interval*);

**XSLT Syntax** <xsl:variable name="result" select="*prefix*:dampen(*tag-string*, *max*, *interval*)"/>

**Release Information** Function introduced in Junos OS Release 9.4.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Prevent the same operation from being repeatedly executed within a script. The **dampen()** function returns **false** if the number of calls to the **jcs:dampen()** function exceeds a ***max*** number of calls in the time interval ***interval***. Otherwise the function returns **true**. The function parameters include an arbitrary string that is used to distinguish different calls to the **jcs:dampen()** function. This tag is stored in the **/var/run** directory on the device.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *interval*—Time interval, in minutes.

*max*—Maximum number of calls to the **jcs:dampen()** function with a given tag allowed before the function returns **false**. This limit is based on the number of calls within a specified time interval.

*tag-string*—Arbitrary string used to distinguish different calls to the **jcs:dampen()** function.

**Return Value** result—Boolean value based on the number of calls to **jcs:dampen()** with a given tag and within a specified time. If the number of calls for a given tag exceeds ***max***, the return value is **false**. If the number of calls is less than ***max***, the return value is **true**.

**Usage Examples** In the following example, if the **jcs:dampen()** function with the tag 'mytag1' is called less than three times in a 10-minute interval, the function returns **true**. If the function is called more than three times within 10 minutes, the function returns **false**.

```
if (jcs:dampen('mytag1', 3, 10)) {
   /* Code for situations when jcs:dampen() with */
   /* the tag 'mytag1' is called less than three times */
   /* within 10 minutes */
} else {
   /* Code for situations when jcs:dampen() with */
   /* the tag 'mytag1' exceeds the three call maximum */
   /* limit within 10 minutes */
}
```

## document() Function (slax Namespace)

**Namespaces** http://xml.libslax.org/slax

**SLAX Syntax** *string* slax:document(*url*, <options>)

**Release Information** Function introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Read data from a file or URL. The data can be encoded in any character set and can be BASE64 encoded. The default character set is "utf-8". Optional arguments specify the character encoding scheme and the encoding format, and define the replacement string for non-XML control characters. Table 19 on page 104 lists the available options.

**Parameters** options—(Optional) Specify the character encoding scheme and format of the data, and define the replacement string for non-XML control characters.

*url*—File or URL from which to read data.

Table 19: Options for slax:document Function

| Option | Description |
|---|---|
| <encoding> *string* | Character encoding scheme. For example "ascii" or "utf-8". |
| <format> *string* | "base64" for BASE64-encoded data. |
| <non-xml> *string* | String used to replace non-XML control characters. If the value is an empty string, non-XML characters are removed. |

**Return Value** *string*—String representing the data.

**Usage Examples**

```
var $data = slax:document($url);

var $options := {
  <encoding> "ascii";
  <format> "base64";
  <non-xml> "#";
}
var $data2 = slax:document($url, $options);
```

## empty( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $result = *prefix*:empty(*node-set* | *string*);

**XSLT Syntax** <xsl:variable name="result" select="*prefix*:empty(*node-set* | *string*)"/>

**Release Information** Function introduced in Junos OS Release 7.6
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Test for the presence of a value and return **true** if the node set or string argument evaluates to an empty value.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** (*node-set* | *string*)—Argument to test for the presence of a value.

**Return Value** result—Boolean value, which is **true** if the argument is empty.

**Usage Examples** In the following example, if **$set** is empty, the script executes the enclosed code block:

```
if ( jcs:empty($set) ) {
  /* Code to handle true value ($set is empty) */
}
```

The following example tests whether the **description** node for interface fe-/0/0/0 is empty. If the description is missing, a **<message>** tag is output.

```
if (jcs:empty(interfaces/interface[name="fe-0/0/0"]/description)) {
  <message> "interface " _name _ " is missing description";
}
```

## evaluate( ) Function (slax Namespace)

**Namespaces** http://xml.libslax.org/slax

**SLAX Syntax** *object* slax:evaluate(*expression*);

**Release Information** Function introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Evaluate a SLAX expression and return the results of the expression. This permits expressions using the extended syntax provided by SLAX in addition to what is allowed in XPath.

**Parameters** *expression*—SLAX expression to evaluate.

**Return Value** *object*—Result of the expression.

**Usage Examples**

```
var $result = slax:evaluate("expr[name == '&']");
```

## execute( ) Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $result = jcs:execute(*connection*, *rpc*);

**XSLT Syntax** <xsl:variable name="result" select="jcs:execute(*connection*, *rpc*)"/>

**Release Information** Function introduced in Junos OS Release 9.3.

**Description** Execute a remote procedure call (RPC) within the context of a specified connection handle. Any number of RPCs may be executed within the context of the connection handle until it is closed with the **jcs:close( )** function.

**Parameters** *connection*—Connection handle generated by a call to the **jcs:open( )** function.

*rpc*—Remote procedure call (RPC) to execute.

**Return Value** result—Results of the executed RPC, which include the contents of the **<rpc-reply>** element, but not the **<rpc-reply>** tag itself. This **$result** variable is the same as that produced by the **jcs:invoke( )** function. By default, the results are in XML format equivalent to the output produced with the **| display xml** option in the CLI.

**Usage Examples** In the following example, the **$rpc** variable is declared and initialized with the Junos XML **<get-interface-information>** element. A call to the **jcs:open( )** function generates a connection handle to the remote device at IP address 10.10.10.1. The user's login and password are provided as arguments to **jcs:open( )** to provide access to the remote device. The code calls **jcs:execute( )** and passes in the connection handle and RPC as arguments. Junos OS on the remote device processes the RPC and returns the results, which are stored in the **$results** variable.

```
var $rpc = <get-interface-information>;
var $connection = jcs:open('10.10.10.1', 'bsmith', 'test123');
var $results = jcs:execute($connection, $rpc);
expr $results;
```

The equivalent XSLT code is:

```
<xsl:variable name="connection" select="jcs:open('10.10.10.1', 'bsmith', 'test123')"/>
<xsl:variable name="rpc">
  <get-interface-information/>
</xsl:variable>
<xsl:variable name="results" select="jcs:execute($connection, $rpc)"/>
<xsl:value-of select="$results"/>
```

## first-of( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $result = *prefix*:first-of(*object*, "*expression*");

**XSLT Syntax** <xsl:variable name="result" select="*prefix*:first-of(*object*, '*expression*')"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Return the first nonempty (non-null) item in a list. If all objects in the list are empty, the default expression is returned. This function provides the same functionality as an **if / else-if / else** construct but in a much more concise format.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *expression*—Default value returned if all objects in the list are empty.

*object*—List of objects.

**Return Value** result—First nonempty (non-null) item in the object list. If all objects in the list are empty, the default expression is returned.

**Usage Examples** In the following example, if the value of **$a** is empty, **$b** is checked. If the value of **$b** is empty, **$c** is checked. If the value of **$c** is empty, **$d** is checked. If the value of **$d** is empty, the string "none" is returned.

```
jcs:first-of($a, $b, $c, $d, "none")
```

In the following example, for each physical interface, the script checks for a description of each logical interface. If a logical interface description does not exist, the function returns the description of the (parent) physical interface. If the parent physical interface description does not exist, the function returns a message that no description was found.

```
var $rpc = <get-interface-information>;
var $results = jcs:invoke($rpc);
for-each ($results/physical-interface/logical-interface) {
  var $description = jcs:first-of(description, ../description, "no description found");
}
```

The equivalent XSLT code is:

```
<xsl:variable name="rpc">
  <get-interface-information/>
</xsl:variable>
<xsl:variable name="results" select="jcs:invoke($rpc)"/>
<xsl:for-each select="$results/physical-interface/logical-interface">
  <xsl:variable name="description"
     select="jcs:first-of(description, ../description, 'no description found')"/>
</xsl:for-each>
```

The code for the **description** variable declaration in the previous examples would be equivalent to the following more verbose **if / else-if / else** construct:

```
var $description = {
  if (description) {
    expr description;
  }
  else if (../description) {
    expr ../description;
  }
  else {
    expr "no description found";
  }
}
```

See also "Example: Displaying DNS Hostname Information Using an Op Script" on page 507.

## get-command( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** *string* = *prefix*:get-command(*string*);

**Release Information** Function introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Prompt the user for input and return the input as a string. If the readline (or libedit) library was found at install time, the return string is entered in the readline history, and will be available using the readline history keystrokes (Ctrl+P and Ctrl+N).

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *string*—Prompt text.

**Return Value** *string*—Command text entered by the user.

**Usage Examples**

```
var $response = slax:get-command("# ");
```

## get-hello( ) Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $capabilities = jcs:get-hello(*connection*);

**XSLT Syntax** <xsl:variable name="capabilities" select="jcs:get-hello(*connection*)"/>

**Release Information** Function introduced in Junos OS Release 11.4.

**Description** Return the session ID and the capabilities of the NETCONF server during a NETCONF session.

During session establishment, the NETCONF server and client application each emit a **<hello>** element to specify which operations, or *capabilities*, they support from among those defined in the NETCONF specification or published as proprietary extensions. The **<hello>** element encloses the **<capabilities>** element and the **<session-id>** element, which specifies the session ID for this NETCONF session.

Within the **<capabilities>** element, a **<capability>** element specifies each supported function. Each capability defined in the NETCONF specification is represented by a uniform resource name (URN). Capabilities defined by individual vendors are represented by uniform resource identifiers (URIs), which can be URNs or URLs.

**Parameters** *connection*—Connection handle generated by a call to the **jcs:open( )** function.

**Return Value** capabilities—XML node set that specifies which operations, or *capabilities*, the NETCONF server supports. The node set also includes the session ID.

**Usage Examples** In the following code snippet, the user, bsmith, establishes a NETCONF session on the default port with the remote device, fivestar, which is running Junos OS. Since the code does not specify a value for the password, the user is prompted for a password during script execution. Once authentication is established, the code calls the **jcs:get-hello()** function and stores the return value in the variable **$hello**, which is then printed to the CLI.

```
var $netconf := {
   <method> "netconf";
   <username> "bsmith";
}
var $connection = jcs:open("fivestar", $netconf);
var $hello = jcs:get-hello($connection);
expr jcs:output($hello);
expr jcs:close($connection);
```

The CLI displays the following output:

```
bsmith@fivestar's password:

urn:ietf:params:xml:ns:netconf:base:1.0
urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
urn:ietf:params:xml:ns:netconf:capability:validate:1.0
urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file
http://xml.juniper.net/netconf/junos/1.0
http://xml.juniper.net/dmi/system/1.0

20847
```

## get-input() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $user-input = *prefix*:get-input(*string*);

**XSLT Syntax** <xsl:variable name="user-input" select="*prefix*:get-input(*string*)"/>

**Release Information** Function introduced in Junos OS Release 9.4.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Invoke a CLI prompt and wait for user input. The user input is defined as a string for subsequent use. If the script is run non-interactively, the function returns an empty value. This function cannot be used with event scripts.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *string*—CLI prompt text.

**Return Value** user-input—Text typed by the user and stored as a string. The return value will be empty if the script is run non-interactively.

**Usage Examples** In the following example, the user is prompted to enter a login name. The user's input is stored in the variable **$username**.

```
var $username = jcs:get-input("Enter login id: ");
```

## get-protocol() Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $protocol = jcs:get-protocol(*connection*);

**XSLT Syntax** <xsl:variable name="protocol" select="jcs:get-protocol(*connection*)"/>

**Release Information** Function introduced in Junos OS Release 11.4.

**Description** Return the session protocol associated with the connection handle. The protocol values are **junoscript**, **netconf**, and **junos-netconf**.

**Parameters** *connection*—Connection handle generated by a call to the **jcs:open()** function.

**Return Value** protocol—Session protocol associated with the connection handle. The values are **junoscript**, **netconf**, and **junos-netconf**.

**Usage Examples** In the following code snippet, the user, bsmith, establishes a NETCONF session on the default port with the remote device, fivestar. Since the code does not specify a value for the password, the user is prompted for a password during script execution. Once authentication is established, the code calls the **jcs:get-protocol()** function and stores the return value in the variable **$protocol**, which is then printed to the CLI.

```
var $netconf := {
   <method> "netconf";
   <username> "bsmith";
}
var $connection = jcs:open("fivestar", $netconf);
var $protocol = jcs:get-protocol($connection);
expr jcs:output($protocol);
expr jcs:close($connection);
```

The CLI displays the following output:

```
bsmith@fivestar's password:

netconf
```

## get-secret() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $user-input = *prefix*:get-secret(*string*);

**XSLT Syntax** <xsl:variable name="user-input" select="*prefix*:get-secret(*string*)"/>

**Release Information** Function introduced in Junos OS Release 9.5R2.

Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Invoke a CLI prompt and wait for user input. Unlike the **jcs:get-input()** function, the input is not echoed back to the user, which makes the function useful for obtaining passwords. The user input is defined as a string for subsequent use. This function cannot be used with event scripts.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *string*—CLI prompt text.

**Return Value** user-input—Text typed by the user and stored as a string.

**Usage Examples** The following example shows how to prompt for a password that is not echoed back to the user:

```
var $password = jcs:get-secret("Enter password: ");
```

## hostname() Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $name = jcs:hostname(*expression*);

**XSLT Syntax** <xsl:variable name="name" select="jcs:hostname(*expression*)"/>

**Release Information** Function introduced in Junos OS Release 9.3.

**Description** Return the fully qualified domain name associated with a given IPv4 or IPv6 address. The DNS server must be configured on the device in order to resolve the domain name.

**Parameters** *expression*—IPv4 or IPv6 address.

**Return Value** name—Hostname associated with the IP address.

**Usage Examples** The following example initializes the variable **address** with the IP address 10.10.10.1. The **$address** variable is passed as the argument to the **jcs:hostname()** function. If the DNS server is configured on the device, the function will resolve the IP address and return the fully qualified domain name, which is stored in the variable **host**.

```
var $address = "10.10.10.1";
var $host = jcs:hostname($address);
```

In XSLT:

```
<xsl:variable name="address" select="10.10.10.1">
<xsl:variable name="host" select="jcs:hostname($address)"/>
```

## invoke() Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $result = jcs:invoke(*rpc*);

**XSLT Syntax** <xsl:variable name="result" select="jcs:invoke(*rpc*)"/>

**Release Information** Function introduced in Junos OS Release 7.6.

**Description** Invoke a remote procedure call (RPC) on the local device. The function is called with one argument, either a string containing a Junos XML API RPC, or a tree containing an RPC. The result contains the contents of the **<rpc-reply>** element, not including the **<rpc-reply>** tag. An RPC allows you to perform functions equivalent to Junos OS operational mode commands.

**Parameters** *rpc*—String containing a Junos XML API RPC or a tree containing an RPC.

**Return Value** result—Results of the executed RPC, which include the contents of the **<rpc-reply>** element, but not the **<rpc-reply>** tag itself. By default, the results are in XML format equivalent to the output produced with the **| display xml** option in the CLI.

**Usage Examples** The following example tests to see if the **interface** argument is included on the command line when the script is executed. If the argument is provided, the output of the **show interfaces terse** operational mode command is narrowed to include only information about the specified interface.

```
<xsl:param name="interface"/>
<xsl:variable name="rpc">
  <get-interface-information>
    <terse/>
    <xsl:if test="$interface">
      <interface-name>
        <xsl:value-of select="$interface"/>
      </interface-name>
    </xsl:if>
  </get-interface-information>
</xsl:variable>
<xsl:variable name="out" select="jcs:invoke($rpc)"/>
```

In this example, the **jcs:invoke()** function calls the Junos XML API RPC **get-software-information**, and stores the unmodified output in the variable **sw**:

```
<xsl:variable name="sw" select="jcs:invoke('get-software-information')"/>
```

## open() Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax**
```
var $connection = jcs:open();
var $connection = jcs:open(remote-hostname, <username>, <passphrase>,
  <routing-instance-name>);
var $connection = jcs:open(remote-hostname, <session-options>);
```

**XSLT Syntax**
```
<xsl:variable name="connection" select="jcs:open()"/>
<xsl:variable name="connection" select="jcs:open(remote-hostname, <username>,
  <passphrase>, <routing-instance-name>)"/>
```

```
<xsl:variable name="connection" select="jcs:open(remote-hostname, <session-options>)"/>
```

**Release Information** Function introduced in Junos OS Release 9.3.
Support for NETCONF sessions added in Junos OS Release 11.4R1.
Support for routing instances added in Junos OS Release 12.2R1.

**Description** Return a connection handle that can be used to execute remote procedure calls (RPCs) using the **jcs:execute()** extension function. To execute an RPC on a remote device, an SSH session must be established. In order for the script to establish the connection, you must either configure the SSH host key information for the remote device on the local device where the script will be executed, or the SSH host key information for the remote device must exist in the known hosts file of the user executing the script.

To redirect the SSH connection to originate from within a specific routing instance, include the routing instance name in the connection parameters. The routing instance must be configured at the **[edit routing-instances]** hierarchy level, and the remote device must be reachable either using the routing table for that routing instance or from one of the interfaces configured under that routing instance.

Starting with Junos OS Release 11.4, the new parameter, ***session-options***, supports the option to create a session either with the Junos XML protocol server on devices running Junos OS or with the NETCONF server on devices where NETCONF service over SSH is enabled. Previously, the function supported only sessions with the Junos XML protocol server on devices running Junos OS.

The connection handle is closed with the **jcs:close()** function.

**Parameters** *passphrase*—(Optional) User's login passphrase. If you do not specify a passphrase and it is required for authentication, you should be prompted for one during script execution by the device to which you are connecting.

*remote-hostname*—Domain name or IP address of the remote router, switch, or security device. If you are opening a local connection, do not pass this value. If you specify a session type, this parameter is required.

*routing-instance-name*—(Optional) Routing instance from within which the SSH connection originates.

*session-options*—(Optional) XML node set that specifies the session protocol and connection parameters. The structure of the node set is:

```
var $session-options := {
    <instance> "routing-instance-name";
    <method> ("junoscript" | "netconf" | "junos-netconf");
    <passphrase> "passphrase";
    <password> "password";
    <port> "port-number";
    <routing-instance> "routing-instance-name";
    <username> "username";
}
```

- **<instance>**—(Optional) Routing instance from within which the SSH connection originates. This element is identical to **<routing-instance>**.
- **<method>**—(Optional) Session protocol. The protocol is one of three values: **junoscript**, **netconf**, or **junos-netconf**. If you do not specify a protocol, a **junoscript** session is created by default. A **<method>** value of **junoscript** establishes a session with the Junos XML protocol server on a device running Junos OS. A **<method>** value of **netconf** establishes a session with a NETCONF server over an SSHv2 connection. A **<method>** value of **junos-netconf** establishes a session with a NETCONF server over an SSHv2 connection on a device running Junos OS.
- **<passphrase> or <password>**—(Optional) User's login passphrase. If you do not specify a passphrase and it is required for authentication, you should be prompted for one during script execution by the device to which you are connecting.
- **<port>**—(Optional) Server port number for **netconf** and **junos-netconf** sessions. For NETCONF sessions, **jcs:open()** connects to the NETCONF server at the default port 830. If you specify a value for **<port>**, **jcs:open()** connects to the given port instead. Specifying a port number has no impact on **junoscript** sessions, which are always established over SSH port 22.
- **<routing-instance>**—(Optional) Routing instance from within which the SSH connection originates. This element is identical to **<instance>**.
- **<username>**—(Optional) User's login name. If you do not specify a username and it is required for the connection, the script uses the local name of the user executing the script.

*username*—(Optional) User's login name. If you do not specify a username and it is required for the connection, the script uses the local name of the user executing the script.

**Return Value** connection—Connection handle to the remote host.

**Usage Examples** The following example shows how to connect to a local device:

```
var $connection = jcs:open();
```

The following example shows how to connect to a remote device:

```
var $connection = jcs:open(remote-hostname);
```

The following example shows how the user, bsmith, with the passphrase "test123" obtains a connection handle to the remote device, fivestar:

```
var $connection = jcs:open("fivestar", "bsmith", "test123");
```

The following example shows how the user, bsmith, with the passphrase "test123" creates a **junos-netconf** session with a device running Junos OS:

```
var $options := {
  <method> "junos-netconf";
  <username> "bsmith";
  <passphrase> "test123";
}
var $connection = jcs:open("fivestar", $options);
```

## output() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:output('*string*', <*string*>);

**XSLT Syntax** <xsl:value-of select="*prefix*:output('*string*', <*string*>)"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Display one or more lines of output text, either to the CLI user (when used in op scripts), or to the output file (when used in event scripts). The function can be called with either a single string argument or with multiple string arguments. Multiple arguments are concatenated into one combined string. A newline terminates the output text.

**jcs:output()** is not supported in commit scripts. Commit scripts use the **<xnm:warning>** and **<xnm:error>** result tree elements to display text to the CLI user.

The behavior of **jcs:output()** differs from the**<output>** result tree element in that **jcs:output()** displays its text immediately, rather than waiting until the conclusion of the script. This makes it suitable for scripts where user interaction is required, such as when the **jcs:get-input()** function is used, or when status messages should be displayed in the midst of script processing. While **jcs:output()** does return a node set, it is always empty and can be ignored. Therefore, the **jcs:output()** function is normally called with the **expr** statement, rather than assigning its result to a variable.

The following escape characters are supported in the output text:

- \\ –Backslash (as of Junos OS Release 10.2)
- \r –Carriage Return
- \" –Double-quote (as of Junos OS Release 10.1R2)
- \n –Newline
- \' – Single-quote
- \t –Tab

Starting with Junos OS Release 10.2, the maximum length for output text is 10 KB, and longer strings are truncated to the supported length.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *string*—Text that is output immediately to the CLI session.

**Usage Examples** SLAX syntax:

```
expr jcs:output('The VPN is up.');
```

XSLT syntax:

```
<xsl:value-of select="jcs:output('The VPN is up.')"/>
```

## parse-ip() Function (jcs Namespace)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

**SLAX Syntax** var $result = jcs:parse-ip("*ipaddress*/(*prefix-length* | *netmask*)");

**XSLT Syntax** <xsl:variable name="result" select="jcs:parse-ip('*ipaddress*/(*prefix-length* | *netmask*')"/>

**Release Information** Function introduced in Junos OS Release 9.0.

**Description** Parse an IPv4 or IPv6 address.

**Parameters** *ipaddress*—IPv4 or IPv6 address.

*netmask*—Netmask defining the network prefix portion of the address.

*prefix-length*—Prefix length defining the number of bits used in the network prefix portion of the address.

**Return Value** result—An array containing:

- Host IP address (or **NULL** in the case of an error)
- Protocol family (inet for IPv4 or inet6 for IPv6)
- Prefix length
- Network address
- Network mask in dotted decimal notation for IPv4 addresses (left blank for IPv6 addresses)

**Usage Examples** In the following examples, an IPv4 address and an IPv6 address are parsed and the resulting output is detailed:

```
var $addr = jcs:parse-ip("10.1.2.10/255.255.255.0");
```

- **$addr[1]** contains the host address **10.1.2.10**.
- **$addr[2]** contains the protocol family **inet**.
- **$addr[3]** contains the prefix length **24**.
- **$addr[4]** contains the network address **10.1.2.0**.
- **$addr[5]** contains the netmask for IPv4 **255.255.255.0**.

```
var $addr = jcs:parse-ip("2001:DB8::c50:8a:800:200C:417A/32");
```

- **$addr[1]** contains the host address **2001:db8:0:c50:8a:800:200c:417a**.
- **$addr[2]** contains the protocol family **inet6**.
- **$addr[3]** contains the prefix length **32**.

- **$addr[4]** contains the network address **2001:db8::**.
- **$addr[5]** is blank for IPv6 (**""**).

## printf( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:printf(*expression*);

**XSLT Syntax** <xsl:value-of select="*prefix*:printf(*expression*)"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Generate formatted output text. Most standard **printf** formats are supported, in addition to some Junos OS–specific formats. The function returns a formatted string but does not print it on call. To use the following Junos OS modifiers, place the modifier between the percent sign (%) and the conversion specifier.

- **j1**—Operator that emits the field only if it changed from the last time the function was called. This assumes that the expression's format string is unchanged.
- **jc**—Operator that capitalizes the first letter of the associated output string.
- **jt{TAG}**—Operator that emits the tag if the associated argument is not empty.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *expression*—Format string containing an arbitrary number of format specifiers and associated arguments to output.

**Usage Examples** In the following example, the **j1** operator suppresses printing the interface identifier so-0/0/0 in the second line of output, because the identifier argument has not changed from the first printing. The **jc** operator capitalizes the output strings **up** and **down**. The **jt{--}** operator does not print the **{--}** tag in the first line of output, because the associated output argument is an empty string. However, the tag is printed in the second line because the associated output is the non-empty string **test**.

```
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ -- }s\n',
        'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/>
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ -- }s\n',
        'so-0/0/0', 'down', 'down', '10.1.2.3', 'test')"/>
```

produces the following output:

```
so-0/0/0         Up    Down  10.1.2.3
                 Down  Down  10.1.2.3 -- test
```

## progress( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0

http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:progress('*string*');

**XSLT Syntax** <xsl:value-of select="*prefix*:progress('*string*')"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Issue a progress message containing the single argument immediately to the CLI session provided that the **detail** flag was specified when the script was invoked.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *string*—Text output to CLI session

**Usage Examples** SLAX syntax:

```
expr jcs:progress('Working...');
```

XSLT syntax:

```
<xsl:value-of select="jcs:progress('Working...')"/>
```

The script must be invoked with the **detail** flag in order for the progress message to appear in the CLI session.

```
user@host> op script1.slax detail
```

```
2010-10-01 16:27:54 PDT: running op script 'script1.slax'
2010-10-01 16:27:54 PDT: opening op script '/var/db/scripts/op/script1.slax'
2010-10-01 16:27:54 PDT: reading op script 'script1.slax'
2010-10-01 16:27:54 PDT: Working...
2010-10-01 16:28:14 PDT: inspecting op output 'script1.slax'
2010-10-01 16:28:14 PDT: finished op script 'script1.slax'
```

## regex( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $result = *prefix*:regex(*pattern*, *string*);

**XSLT Syntax** <xsl:variable name="result" select="*prefix*:regex(*pattern*, *string*)"/>

**Release Information** Function introduced in Junos OS Release 7.6
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Evaluate a regular expression against a given string argument and return any matches. This function requires two arguments: the regular expression and the string to which the regular expression is compared.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *pattern*—Regular expression that is evaluated against the string argument.

*string*—String within which to search for matches of the specified regular expression.

**Return Value** result—Array of strings that match the given regex pattern within the string argument.

**Usage Examples** In the following example, the regex pattern consists of four distinct groups. The first group consists of the entire expression. The three subsequent groups are each of the parentheses–enclosed expressions within the main expression. The results for each **jcs:regex()** function call contain an array of the matches of the regex pattern to each of the specified strings.

```
var $pattern = "([0-9]+)(:*)([a-z]*)";
var $a = jcs:regex($pattern, "123:xyz");
var $b = jcs:regex($pattern, "r2d2");
var $c = jcs:regex($pattern, "test999!!!");

$a[1] == "123:xyz"   # string that matches the full reg expression
$a[2] == "123"       # ([0-9]+)
$a[3] == ":"         # (:*)
$a[4] == "xyz"       # ([a-z]*)
$b[1] == "2d"        # string that matches the full reg expression
$b[2] == "2"         # ([0-9]+)
$b[3] == ""          # (:*)  [empty match]
$b[4] == "d"         # ([a-z]*)
$c[1] == "999"       # string that matches the full reg expression
$c[2] == "999"       # ([0-9]+)
$c[3] == ""          # (:*)  [empty match]
$c[4] == ""          # ([a-z]*)  [empty match]
```

## sleep() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:sleep(*seconds*, <*milliseconds*>);

**XSLT Syntax** <xsl:value-of select="*prefix*:sleep(*seconds*, <*milliseconds*>)"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Cause the script to pause for a specified number of seconds and (optionally) milliseconds. You can use this function to help determine how a device component works over time. To do this, write a script that issues a command, calls the **jcs:sleep()** function, and then reissues the same command.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *milliseconds*—(Optional) Number of milliseconds the script should sleep.

*seconds*—Number of seconds the script should sleep.

**Usage Examples** In the following example, **jcs:sleep(1)** causes the script to sleep for 1 second, and **jcs:sleep(0, 10)** causes the script to sleep for 10 milliseconds:

SLAX syntax:

```
expr jcs:sleep(1);
expr jcs:sleep(0, 10);
```

XSLT syntax:

```
<xsl:value-of select="jcs:sleep(1)"/>
<xsl:value-of select="jcs:sleep(0, 10)"/>
```

## split( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $substrings = *prefix*:split(*expression*, *string*, <*limit*>);

**XSLT Syntax** <xsl:variable name="substrings" select="*prefix*:split(*expression*, *string*, <*limit*>)"/>

**Release Information** Function introduced in Junos OS Release 8.4
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Split a string into an array of substrings delimited by a regular expression pattern. If the optional integer argument ***limit*** is specified, the function splits the entire string into ***limit*** number of substrings. If there are more than ***limit*** number of matches, the substrings include the first ***limit***-1 matches as well as the remaining portion of the original string for the last match.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *expression*—Regular expression pattern used as the delimiter.

*limit*—(Optional) Number of substrings into which to break the original string.

*string*—Original string.

**Return Value** $substrings—Array of ***limit*** number of substrings. If ***limit*** is not specified, the result array size is equal to the number of substrings extracted from the original string as determined by the specified delimiter.

**Usage Examples** In the following example, the original string is "123:abc:456:xyz:789". The **jcs:split()** function breaks this string into substrings that are delimited by the regular expression pattern, which in this case is a colon(:). The optional parameter ***limit*** is not specified, so the function returns an array containing all the substrings that are bounded by the delimiter(:).

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789");
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc"
$substrings[3] == "456"
$substrings[4] == "xyz"
$substrings[5] == "789"
```

The following example uses the same original string and regular expression as the previous example, but in this case, the optional parameter ***limit*** is included. Specifying ***limit***=2 causes the function to return an array containing only two substrings. The substrings include the first match, which is "123" (the same first match as in the previous example), and a second match, which is the remaining portion of the original string after the first occurrence of the delimiter.

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789", 2);
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc:456:xyz:789"
```

## sysctl() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** var $value = *prefix*:sysctl(*sysctl-value*, "(i | s)");

**XSLT Syntax** <xsl:variable name="value" select="*prefix*:sysctl(*sysctl-value*, '(i | s)')"/>

**Release Information** Function introduced in Junos OS Release 7.6
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Return the given **sysctl** value as a string or an integer. Use the "**i**" argument to specify an integer. Use the "**s**" argument to specify a string.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *sysctl-value*—**sysctl** value to convert to a string or integer.

**Return Value** $value—Returned string or integer value.

**Usage Examples**

```
var $value = jcs:sysctl("kern.hostname", "s");
```

## syslog( ) Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:syslog(*priority*, *message*, <*message2*>, <*message3*> ...);

**XSLT Syntax** <xsl:value-of select="*prefix*:syslog(*priority*, *messsage*, <*message2*>, <*message3*>)"/>

**Release Information** Function introduced in Junos OS Release 7.6
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Log messages with the specified priority to the system log file. The priority can be expressed as a ***facility.severity*** string or as a calculated integer. The ***message*** argument is a string or variable that is written to the system log file. Optionally, additional strings or variables can be included in the argument list. The ***message*** argument is concatenated with any additional message arguments, and the concatenated string is written to the system log file. The syslog file is specified at the **[edit system syslog]** hierarchy level of the configuration.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *message*—String or variable that is output to the system log file.

*message2*—(Optional) Any additional number of strings or variable names passed as arguments to the function. These are concatenated with the ***message*** argument and output to the system log file.

*priority*—Priority given to the syslog message. The priority can be specified as a ***facility.severity*** string, or it can expressed as an integer calculated from the corresponding numeric values of the facility and severity strings. Table 20 on page 122 and Table 21 on page 123 show the facility and severity strings available and their corresponding numeric values.

The integer value of the ***priority*** parameter is calculated by multiplying the facility string numeric value by 8 and adding the severity string numeric value. For example, if the ***facility.severity*** string pair is **"pfe.alert"**, the priority value is 161 ((20 x 8)+1).

### Table 20: Facility Strings

| Facility String | Description | Numeric Value |
|---|---|---|
| auth | Authorization system | 4 |
| change | Configuration change log | 22 |
| conflict | Configuration conflict log | 21 |
| daemon | Various system processes | 3 |
| external | Local external applications | 18 |

Table 20: Facility Strings *(continued)*

| Facility String | Description | Numeric Value |
|---|---|---|
| firewall | Firewall filtering system | 19 |
| ftp | FTP processes | 11 |
| interact | Commands executed by the UI | 23 |
| pfe | Packet Forwarding Engine | 20 |
| user | User processes | 1 |

Table 21: Severity Strings

| Severity String | Description | Numeric Value |
|---|---|---|
| alert | Conditions that should be corrected immediately | 1 |
| crit | Critical conditions | 2 |
| debug | Debug messages | 7 |
| emerg or panic | Panic conditions | 0 |
| err or error | Error conditions | 3 |
| info | Informational messages | 6 |
| notice | Conditions that should be specially handled | 5 |
| warn or warning | Warning messages | 4 |

**Usage Examples**

The following three examples log **pfe** messages with an **alert** priority. The string "**mymessage**" is output to the system log file. All three examples are equivalent.

```
expr jcs:syslog("pfe.alert", "mymessage");

expr jcs:syslog(161, "mymessage");

var $message = "mymessage";
expr jcs:syslog("pfe.alert", $message);
```

The following example logs **pfe** messages with an **alert** priority similar to the previous example. In this example, however, there are additional string arguments. For this case, the concatenated string "**mymessage mymessage2**" is output to the system log file.

```
expr jcs:syslog("pfe.alert", "mymessage ", "mymessage2");
```

### trace() Function (jcs and slax Namespaces)

**Namespaces** http://xml.juniper.net/junos/commit-scripts/1.0
http://xml.libslax.org/slax

**SLAX Syntax** expr *prefix*:trace('*expression*');

**XSLT Syntax** <xsl:value-of select="*prefix*:trace('*expression*')"/>

**Release Information** Function introduced in Junos OS Release 7.6.
Support for the slax namespace http://xml.libslax.org/slax added in Junos OS Release 12.2.

**Description** Issue a trace message, which is sent to the trace file. You must configure **traceoptions** under the respective script type in the configuration hierarchy in order to output a message to the trace file using the **jcs:trace()** function. The output goes to the configured trace file. If **traceoptions** is enabled, but no trace file is explicitly configured, the output goes to the default trace file for that script type.

The *prefix* associated with the namespace URI should be defined in the prefix-to-namespace mapping in the style sheet.

**Parameters** *expression*—String that is output to the trace file.

**Usage Examples** SLAX syntax:

```
expr jcs:trace('test');
```

XSLT syntax:

```
<xsl:value-of select="jcs:trace('test')"/>
```

## Junos Script Automation: Named Templates in the jcs Namespace Overview

Junos OS provides several named templates to make scripting tasks easier in commit, op, and event scripts. The named templates reside in the **junos.xsl** import file, which is included with the standard Junos OS installation available on all switches, routers, and security devices running Junos OS.

The Junos OS named templates are defined in the namespace with the associated Uniform Resource Identifier (URI) http://xml.juniper.net/junos/commit-scripts/1.0 . The templates use the **jcs:** prefix to avoid conflicting with standard XSLT templates or user–defined templates of the same name in a script. To use the Junos named templates in a script, you must include the namespace URI in your style sheet declaration. Map the **jcs** prefix to the URI by including the **xmlns:jcs** attribute in the opening **<xsl:stylesheet>** tag element for XSLT scripts or by including the **ns jcs** statement in SLAX scripts. You must also import the **junos.xsl** file into the script by including the **<xsl:import/>** tag element in XSLT scripts or the **import** statement in SLAX scripts and specifying the **junos.xsl** file location.

To call a named template in a script, include the **<xsl:call-template name="*template-name*">** element for XSLT scripts or the **call** statement for SLAX scripts and pass along any required or optional parameters. Template parameters are assigned by name and can appear in any order. This differs from functions where the arguments must be passed into the function in the precise order specified by the function definition.

The following example imports the **junos.xsl** file into a script and maps the **jcs** prefix to the namespace identified by the URI http://xml.juniper.net/junos/commit-scripts/1.0. The script demonstrates a call to the **jcs:edit-path** template.

**XSLT Syntax**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
        xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>
    ...
  <xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]">
    <xnm:warning>
      <xsl:call-template name="jcs:edit-path"/>
      <message>interface configured</message>
    </xnm:warning>
  </xsl:for-each>
    ...
</xsl: stylesheet>
```

**SLAX Syntax**

```
version 1.0;
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
    ...
  for-each ( interfaces/interface[starts-with(name,'so-') ) {
    <xnm:warning> {
      call jcs:edit-path();
      <message> “interface configured”;
    }
  }
    ...
```

For more information about attributes and tag elements to include in your scripts, see “Required Boilerplate for Commit Scripts” on page 270, “Required Boilerplate for Op Scripts” on page 463, and “Required Boilerplate for Event Scripts” on page 665.

**Related Documentation**

- Summary of Extension Functions in the jcs and slax Namespaces on page 97
- Junos Named Templates in the jcs Namespace Summary on page 125
- Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces on page 95
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137

## Junos Named Templates in the jcs Namespace Summary

The Junos named templates are summarized in the following table:

Table 22: Junos Named Templates

| Template | Description |
|---|---|
| jcs:edit-path | Generate an **<edit-path>** element suitable for inclusion in an **<xnm:error>** or **<xnm:warning>** element. |
| jcs:emit-change | Generate a **<change>** or **<transient-change>** element, which results in a persistent or transient change to the configuration. |
| jcs:emit-comment | Emit a simple comment that indicates a change was made by a commit script. |
| jcs:grep | Search a file for all instances matching a specified regular expression and write the matching strings and corresponding lines to the result tree. |
| jcs:load-configuration | Make structured changes to the Junos OS configuration using an op script. |
| jcs:statement | Generate a **<statement>** element suitable for inclusion in an **<xnm:error>** or **<xnm:warning>** element. |

**Related Documentation**

- Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces on page 95
- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137

## Junos Named Templates in the jcs Namespace

The templates are discussed in more detail in the following sections:


- jcs:edit-path Template on page 126
- jcs:emit-change Template on page 127
- jcs:emit-comment Template on page 130
- jcs:grep Template on page 131
- jcs:load-configuration Template on page 131
- jcs:statement Template on page 134


### jcs:edit-path Template

**XSLT Syntax**

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

**SLAX Syntax**

```
call jcs:edit-path($dot=expression);
```

**Description** Generate an **<edit-path>** element suitable for inclusion in an **<xnm:error>** or **<xnm:warning>** element. This template converts a location in the configuration hierarchy into the standard text representation that you would see in the Junos OS configuration mode banner. By default, the location of the configuration error is passed into the **jcs:edit-path** template as the value of **dot**. This location defaults to " **.** ", the current position in the XML hierarchy. You can alter the default by including a valid XPath expression for the **dot** parameter when you call the template.

**Parameters** dot—XPath expression specifying the hierarchy level. The default location is the position in the XML hierarchy that the script is currently evaluating. You can alter the default when you call the template by including a valid XPath expression either for the **dot** parameter in SLAX scripts or for the **select** attribute of the **dot** parameter in XSLT scripts.

**Usage Examples** The following example demonstrates how to call the **jcs:edit-path** template in a commit script and set the context to the **[edit chassis]** hierarchy level:

```
<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>
```

When you commit a configuration that does not enable IP source routing, the code generates an **<xnm:warning>** element, which results in the following command-line interface (CLI) output:

```
user@host# commit
[edit chassis] # The hierarchy level is generated by the jcs:edit-path template.
    warning: IP source-route processing is not enabled.
commit complete
```

## jcs:emit-change Template

**XSLT Syntax**

```
<xsl:call-template name="jcs:emit-change">
  <xsl:with-param name="content">
    ...
  </xsl:with-param>
  <xsl:with-param name="dot" select="expression"/>
  <xsl:with-param name="message">
    <xsl:text>message</xsl:text>
  </xsl:with-param>
  <xsl:with-param name="name" select="name($dot)"/>
  <xsl:with-param name="tag" select="'(change | transient-change)'"/>
</xsl:call-template>
```

**SLAX Syntax**

```
call jcs:emit-change($dot=expression, $name = name($dot), $tag = "(change |
transient-change)" {
  with $content = {
    ...
```

```
    }
    with $message = {
      expr "message";
    }
  }
```

**Description** Generate a **<change>** or **<transient-change>** element, which results in a persistent or transient change to the configuration.

**Parameters** This template includes the following optional parameters:

content—Content of the persistent or transient change, relative to **dot**.

dot—XPath expression specifying the hierarchy level at which the change will be made. The default location is the position in the XML hierarchy that the script is currently evaluating. You can alter the default when you call the template by including a valid XPath expression either for the **dot** parameter in SLAX scripts or for the **select** attribute of the **dot** parameter in XSLT scripts.

message—Warning message displayed in the CLI notifying the user that the configuration has been changed. The message parameter automatically includes the edit path, which defaults to the current location in the XML hierarchy. To change the default edit path, specify a valid XPath expression either for the **dot** parameter in SLAX scripts or for the **select** attribute of the **dot** parameter in XSLT scripts.

name—Allows you to refer to the current element or attribute. The **name()** XPath function returns the name of an element or attribute. The **name** parameter defaults to the value **name($dot)**, which is the name of the element in **dot** (which in turn defaults to " **.** ", which is the current element).

tag—Type of change to generate. By default, the **jcs:emit-change** template generates a persistent change, as designated by the **'change'** expression. To specify a transient change, you must include the **tag** parameter and include the **'transient-change'** expression.

**Usage Examples** The following example demonstrates how to call the **jcs:emit-change** template in a commit script:

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

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

When you commit a configuration that includes one or more interfaces that have IS-IS enabled but do not have the **family mpls** statement included at the **[edit interfaces *interface-name* unit *logical-unit-number*]** hierarchy level, the **jcs:emit-change** template adds the **family mpls** statement to the configuration and generates the following CLI output:

```
[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
```

The **content** parameter of the **jcs:emit-change** template provides a simpler method for specifying a change to the configuration. For example, consider the following code:

```
<xsl:with-param name="content">
  <family>
    <mpls/>
  </family>
</xsl:with-param>
```

The **jcs:emit-change** template converts the **content** parameter into a **<change>** request. The **<change>** request inserts the provided partial configuration content into the complete hierarchy of the current context node. Thus, the **jcs:emit-change** template changes the hierarchy information in the **content** parameter into the following code:

```
<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>
```

If a transient change is required, the **tag** parameter can be passed in as '**transient-change**', as shown here:

```
<xsl:with-param name="tag" select="'transient-change'"/>
```

The extra quotation marks are required to allow XSLT to distinguish between the string "**transient-change**" and the contents of a node named "**transient-change**". If the change is relative to a node other than the context node, the parameter **dot**can be set to that node, as shown in the following example, where context is set to the **[edit chassis]** hierarchy level:

```
<xsl:for-each select="interfaces/interface/unit">
  ...
  <xsl:call-template name="jcs:emit-change">
    <xsl:with-param name="dot" select="chassis"/>
      ...
```

See also "Example: Imposing a Minimum MTU Setting" on page 395.

## jcs:emit-comment Template

**XSLT Syntax**

```
<junos:comment>
  <xsl:text>...</xsl:text>
</junos:comment>
```

**Description** Emit a simple comment that indicates a change was made by a commit script. The template contains a **<junos:comment>** element. You never call the **jcs:emit-comment** template directly. Rather, you include its **<junos:comment>** element and the child element **<xsl:text>** inside a call to the **jcs:emit-change** template, a **<change>** element, or a **<transient-change>** element.

**Usage Examples** The following example demonstrates how to call this template in a commit script:

```
<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>
```

When you issue the **show firewall** configuration mode command, the following output appears:

```
[edit]

user@host# show firewall
family inet {
    term very-last {
          /* This term was added by a commit script */
    then accept;
    }
}
```

## jcs:grep Template

**XSLT Syntax**

```
<xsl:call-template name="jcs:grep">
  <xsl:with-param name="filename" select="filename"/>
  <xsl:with-param name="pattern" select="pattern"/>
</xsl:call-template>
```

**SLAX Syntax**

```
call jcs:grep($filename=filename, $pattern=pattern);
```

**Description** Search the given input file for all instances matching the specified regular expression and write the matching strings and corresponding lines to the result tree. The pattern is matched to each line of the file. The template does not support matching a pattern spanning multiple lines.

If the regular expression contains a syntax error, the template generates an error for every line of the file. For each match, the template adds a **<match>** element, which contains **<input>** and **<output>** child tags, to the result tree. The template writes the matching string to the **<output>** element and writes the corresponding matching line to the **<input>** element.

```
<match> {
  <input>
  <output>
}
```

Starting with Junos OS Release 11.1, if an absolute path is not specified for the input file, the default path is relative to the user's home directory for op scripts, and it is relative to the **/var/tmp/** directory for commit scripts and for event scripts that are enabled at the **[edit event-options event-script]** hierarchy level. For event scripts that are enabled at the **[edit system scripts]** hierarchy level, the default path is relative to the top-level directory, */*.

**Parameters** filename—(Mandatory) Absolute or relative path and filename of the file to search.

Starting with Junos OS Release 11.1, if you do not specify an absolute path, the path is relative to the user's home directory for op scripts, and it is relative to the **/var/tmp/** directory for commit scripts and for event scripts that are enabled at the **[edit event-options event-script]** hierarchy level. For event scripts that are enabled at the **[edit system scripts]** hierarchy level, the default path is relative to the top-level directory, */*.

pattern—(Mandatory) Regular expression.

**Usage Examples** "Example: Searching Files Using an Op Script" on page 531

## jcs:load-configuration Template

**XSLT Syntax**

```
<xsl:call-template name="jcs:load-configuration">
  <xsl:with-param name="action" select="(merge | override | replace)"/>
  <xsl:with-param name="commit-options" select="node-set"/>
  <xsl:with-param name="configuration" select="configuration-data"/>
  <xsl:with-param name="connection" select="connection-handle"/>
```

```
    <xsl:with-param name="rollback" select="number"/>
</xsl:call-template>
```

**SLAX Syntax**

```
call jcs:load-configuration($action="(merge | override | replace)",
$commit-options=node-set, $configuration=configuration-data,
$connection=connection-handle, $rollback=number);
```

**Description** Make structured changes to the Junos OS configuration using an op script. When called, the template locks the configuration database, loads the configuration changes, commits the configuration, and then unlocks the configuration database.

The **jcs:load-configuration** template makes changes to the configuration in **configure exclusive** mode. In this mode, Junos OS locks the candidate *global* configuration for as long as the script accesses the shared database and makes changes to the configuration without interference from other users.

If another user is currently editing the configuration in **configure exclusive** mode or if the database is already locked when the template is called, the call fails. In addition, if there are existing, uncommitted changes to the configuration when the template is called, the commit will fail. If the template call is successful but the commit fails, Junos OS discards the uncommitted changes and rolls back the configuration.

**Parameters** action—Specifies how to load the configuration changes with respect to the candidate configuration. The following options are supported:

- merge—Combine the candidate configuration and the incoming configuration changes. If the candidate configuration and the incoming configuration contain conflicting statements, the incoming statements override those in the candidate configuration.
- override—Replace the entire candidate configuration.
- replace—Replace existing statements in the candidate configuration with the tags of the same name that are marked with **replace:** in the incoming configuration. If there is no existing statement of the same name in the candidate configuration, the statement is added to the candidate configuration.

commit-options—Node set defining options that customize the commit command. The default value is null. Supported commit options are:

- check—Check the correctness of the candidate configuration syntax, but do not commit the changes.
- force-synchronize—Force the commit on the other Routing Engine (ignore any warnings).
- log—Write the specified message to the commit log. This is identical to the CLI configuration mode command **commit comment**.
- synchronize—Synchronize the commit on both Routing Engines.

configuration—XML configuration changes. The configuration changes are incorporated into the candidate configuration as specified by the action parameter. Starting with Junos OS Release 12.2, you can also supply a NULL configuration. If the configuration

data value is NULL, the template performs a simple commit of the candidate configuration.

connection—Connection handle generated by a call to the **jcs:open()** function.

rollback—Return to a previously committed configuration. Specify the rollback number of the configuration, and the configuration you specify is loaded from the associated file. The software saves the last 50 committed configurations. The rollback parameter is available starting with Junos OS Release 12.2.

**Usage Examples** The following example calls the **jcs:load-configuration** template to modify the configuration to disable an interface. The interface name is supplied by the user and stored in the variable **interface-name**. All of the values required for the **jcs:load-configuration** template are defined as variables, which are then passed into the template as arguments.

In this example, the configuration data that includes the changes to the configuration are stored in the variable **disable**. This is the value used for the **configuration** parameter of the **jcs:load-configuration** template. The **load-action** variable is initialized to **merge**, which merges the configuration changes in the **disable** variable with the candidate configuration. This is the equivalent of the CLI configuration mode command **load merge**.

The **options** variable uses the **:=** operator to create a node-set, which is passed to the template as the value of the **commit-options** parameter. This example uses the **synchronize** commit option . If the commit succeeds, it will commit the configuration changes on both Routing Engines. The **log** tag is also included to add the description of the commit to the commit log file for future reference.

The call to the **jcs:open()** function opens a connection with the Junos OS management process (mgd) and returns a connection handle that is stored in the **conn** variable. All of the defined variables are passed as arguments to the **jcs:load-configuration** template at the time that it is called.

SLAX syntax:

```
var $disable = {
  <configuration> {
    <interfaces> {
      <interface> {
        <name> $interface-name;
        <disable>;
      }
    }
  }
}
var $load-action = "merge";
var $options := {
  <commit-options> {
    <synchronize>;
    <log> "disabling interface on both routing engines";
}
var $conn = jcs:open();
```

```
var $disable-results := {
  call jcs:load-configuration($action=$load-action, $commit-options=$options,
      $configuration = $disable, $connection = $conn);
}
if ($disable-results//xnm:error) {
  for-each ($disable-results//xnm:error) {
    <output> message;
  }
}
var $close-results = jcs:close($conn);
```

The **:=** operator copies the results of the **jcs:load-configuration** template call to a temporary variable and runs the **node-set** function on that variable. The **:=** operator ensures that the **disable-results** variable is a node-set rather than a result tree fragment so that the script can access the contents. The **if** code block is included to output any error messages that may indicate a problem in committing the configuration. The **jcs:close** function closes the connection.

In XSLT, the code corresponding to the SLAX call to **jcs:load-configuration** template is:

```
<xsl:variable name="disable-results-temp">
  <xsl:call-template name="jcs:load-configuration">
    <xsl:with-param name="action" select="$load-action"/>
    <xsl:with-param name="commit-options" select="$options"/>
    <xsl:with-param name="configuration" select="$disable"/>
    <xsl:with-param name="connection" select="$conn"/>
  </xsl:call-template>
</xsl:variable>

<xsl:variable xmlns ext="http:xmlsoft.org/XSLT/namespace" \
  name="disable-results" select="ext:node-set($disable-results-temp)"/>
```

## jcs:statement Template

**XSLT Syntax**

```
<xsl:call-template name="jcs:statement">
  <xsl:with-param name="dot" select="expression"/>
</xsl:call-template>
```

**SLAX Syntax**

```
call jcs:statement($dot=expression);
```

**Description** Generate a **<statement>** element suitable for inclusion in an **<xnm:error>** or **<xnm:warning>** element. This location defaults to " **.** ", the current position in the XML hierarchy. If the error is not at the current position in the XML hierarchy, you can alter the default when you call the template by including a valid XPath expression either for the **dot** parameter in SLAX scripts or for the **select** attribute of the **dot** parameter in XSLT scripts.

**Parameters** dot—XPath expression specifying the hierarchy level. The default location is the position in the XML hierarchy that the script is currently evaluating. You can alter the default when you call the template by including a valid XPath expression either for the **dot** parameter in SLAX scripts or for the **select** attribute of the **dot** parameter in XSLT scripts.

**Usage Examples** The following example demonstrates how to call the **jcs:statement** template in a commit script:

```
<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>
```

When you commit a configuration that includes a SONET/SDH interface with a maximum transmission unit (MTU) setting less than a specified minimum, the **<xnm:error>** element results in the following CLI output:

```
[edit]

user@host# commit
[edit interfaces interface so-1/2/3]
    'mtu 576;'  # mtu statement generated by the jcs:statement template
    SONET interfaces must have a minimum MTU of 2048.
error: 1 error reported by commit scripts
error: commit script failure
```

The test of the MTU setting is not performed in the **<xnm:error>** element. For the full example, see "Example: Imposing a Minimum MTU Setting" on page 395.

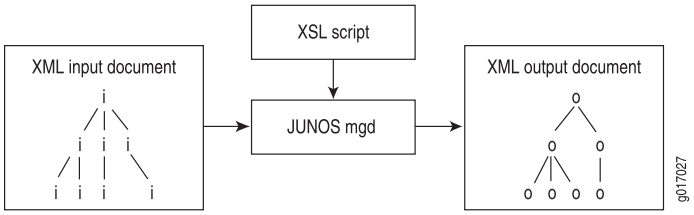
## xsl:template match="/" Template

**Syntax** <xsl:template match="/">

**Description** The **<xsl:template match="/">** template is an unnamed template in the **junos.xsl** file that allows you to use shortened XPath expressions in commit scripts. You must import the **junos.xsl** file to use this template. However, because this template is not in the **jcs** namespace, you do not need to map to the **jcs** namespace in your style sheet declaration in order to use this template.

Junos OS provides XML-formatted input to a script. Commit script input consists of an XML representation of the post-inheritance candidate configuration file. When you execute a script, the Junos OS management process (mgd) generates an XML-formatted output document as the product of its evaluation of the input document, as shown in Figure 3 on page 135.

### Figure 3: Commit Script Input and Output



Generally, an XSLT engine uses recursion to evaluate the entire input document. However, the **<xsl:apply-templates>** instruction allows you to limit the scope of the evaluation so that the management process (the Junos OS's XSLT engine) must evaluate only a subset of the input document.

The **<xsl:template match="/">** template is an unnamed template that uses the **<xsl:apply-templates>** instruction to specify the contents of the input document's **<configuration>** element as the only node to be evaluated in the generation of the output document.

The **<xsl:template match="/">** template contains the following tags:

```
1  <xsl:template match="/">
2    <commit-script-results>
3      <xsl:apply-templates select="commit-script-input/configuration"/>
4    </commit-script-results>
5  </xsl:template>
```

Line 1 matches the root node of the input document. When the management process sees the root node of the input document, this template is applied.

```
1  <xsl:template match="/">
```

Line 2 designates the root, top-level tag of the output document. Thus, Line 2 specifies that the evaluation of the input document results in an output document whose top-level tag is **<commit-script-results>**.

```
2    <commit-script-results>
```

Line 3 limits the scope of the evaluation of the input document to the contents of the **<configuration>** element, which is a child of the **<commit-script-input>** element.

```
3      <xsl:apply-templates select="commit-script-input/configuration"/>
```

Lines 4 and 5 are closing tags.

You do not need to explicitly include the **<xsl:template match="/">** template in your scripts because this template is included in the import file **junos.xsl**.

When the **<xsl:template match="/">** template executes the **<xsl:apply-templates>** instruction, the script jumps to a template that matches the **<configuration>** tag. This template, **<xsl:template match="configuration">**, is part of the commit script boilerplate that you must include in all of your commit scripts:

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

Thus, the import file **junos.xsl** contains a template that points to a template explicitly referenced in your script.

**Usage Examples**

The following example contains the **<xsl:if>** programming instruction and the **<xnm:warning>** element. The logical result of both templates is:

```
<commit-script-results>      <!-- from template in junos.xsl import file -->
  <xsl:if test="not(system/host-name)"> <!-- from "configuration" template -->
    <xnm:warning xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
```

```
        <edit-path>[edit system]</edit-path>
        <statement>host-name</statement>
        <message>Missing a hostname for this device.</message>
      </xnm:warning>
    </xsl:if>   <!-- end of "configuration" template -->
  </commit-script-results>   <!-- end of template in junos.xsl import file -->
```

When you import the **junos.xsl** file and explicitly include the **<xsl:template match="configuration">** tag in your commit script, the context (**dot**) moves to the **<configuration>** node. This allows you to write all XPath expressions relative to that point. This technique allows you to simplify the XPath expressions you use in your commit scripts. For example, instead of writing this, which matches the device with hostname **atlanta**:

```
<xsl:if test="starts-with(commit-script-input/configuration/system/host-name,
'atlanta')">
```

You can write this:

```
<xsl:if test="starts-with(system/host-name, 'atlanta')">
```

**Related Documentation**

- Junos Named Templates in the jcs Namespace Summary on page 125
- Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces on page 95
- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137
- **apply-templates on page 165**
- **xsl:apply-templates on page 149**
- **xsl:template on page 157**

## Junos Script Automation: Global Parameters and Variables in the junos.xsl File

The **junos.xsl** import file declares several predefined parameters and a global variable of type node-set, which provide information about the Junos OS environment that is useful for creating scripts that respond to a variety of complex scenarios. The global parameters and variable are available for use in any commit, op, or event script that imports the **junos.xsl** file.

To use the parameters or variable in a script, you must import the **junos.xsl** file by including the **<xsl:import>** tag in the style sheet declaration of an XSLT script or by including the **import** statement in a SLAX script and specifying the **junos.xsl** file location as shown in the following sample code:

**XSLT Syntax**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0">
  <xsl:import href="../import/junos.xsl"/>
  ...
</xsl: stylesheet>
```

**SLAX Syntax**

```
version 1.0;
import "../import/junos.xsl";
```

The default arguments are described in detail in the following sections:

- Global Parameters on page 138
- Global Variable on page 139

## Global Parameters

Several predefined global parameters are available for use in commit, op, and event scripts. The parameters provide information about the Junos OS environment. Table 23 on page 138 describes the built-in arguments.

**Table 23: Predefined Parameters Available to Automation Scripts**

| Name | Description | Example |
|---|---|---|
| $hostname | Hostname of the local device | Tokyo |
| $localtime | Local time when the script is executed | Fri Dec 10 11:42:21 2010 |
| $localtime-iso | Local time, in ISO format, when the script is executed | 2010-12-10 11:42:21 PST |
| $product | Model of the local device | m10i |
| $script | Filename of the executing script | **test.slax** |
| $user | Local name of the user executing the script | root |

The predefined global parameters are declared in the **junos.xsl** file. You do not need to declare these parameters in a script in order to use them. Access the value of the global parameters in a script by prefixing the parameter name with the dollar sign ($), as shown in the following example:

SLAX syntax:

```
if ($user != "root") {
   var $script-message = $user _ " does not have permission to execute " _ $script;
   expr jcs:output($script-message);
}
```

XSLT syntax:

```
<xsl:if test="$user != 'root'">
   <xsl:variable name="script-message"
         select="concat($user, ' does not have permission to execute ', $script)"/>
   <xsl:value-of select="jcs:output($script-message)"/>
</xsl:if>
```

## Global Variable

Starting with Junos OS Release 11.1, Junos OS also provides a single global variable, **$junos-context**, which is accessible for use in all commit, op, or event scripts that import the **junos.xsl** file. The **$junos-context** variable is a node-set, which has elements that mirror the original global parameters described in "Global Parameters" on page 138 as well as additional elements with information about the Junos OS environment, such as whether a script is executed on the master Routing Engine.

The **$junos-context** variable contains the **<junos-context>** node and the following hierarchy, which is common to and embedded in the source tree of all scripts:

```
<junos-context>
  <chassis></chassis>
  <hostname></hostname>
  <localtime></localtime>
  <localtime-iso></localtime-iso>
  <pid></pid>
  <product></product>
  <re-master/>
  <routing-engine-name></routing-engine-name>
  <script-type></script-type>
  <tty></tty>
  <user-context>
    <class-name></class-name>
    <login-name></login-name>
    <uid></uid>
    <user></user>
  </user-context>
</junos-context>
```

Additionally, script-specific information is available depending on the type of script executed. For op scripts, the **<op-context>** element is also included in the source tree provided to an op script:

```
<junos-context>
  <op-context>
    <via-url/>
  </op-context>
</junos-context>
```

For commit scripts, the **<commit-context>** element is also included in the source tree provided to a commit script:

```
<junos-context>
  <commit-context>
    <commit-comment>"This is a test commit"</commit-comment>
    <commit-boot/>
    <commit-check/>
    <commit-sync/>
    <commit-confirm/>
    <database-path/>
  </commit-context>
</junos-context>
```

Table 24 on page 140 identifies each node of the **$junos-context** variable node-set, provides a brief description of the node, and gives examples of values for any elements that are not input to a script as an empty tag.

### Table 24: Global Variable $junos-context Available to Automation Scripts

| Parent Node | Node | Description | Example content |
|---|---|---|---|
| <junos-context> | <chassis> | Specifies whether the script is executed on a component of a routing matrix, the Root System Domain (RSD), or a Protected System Domain (PSD) | scc, lcc (TX Matrix)<br>psd, rsd (JCS)<br>others |
| | <hostname> | Hostname of the local device | Tokyo |
| | <localtime> | Local time when the script is executed | Fri Dec 10 11:42:21 2010 |
| | <localtime-iso> | Local time, in ISO format, when the script is executed | 2010-12-10 11:42:21 PST |
| | <pid> | cscript process ID | 5257 |
| | <product> | Model of the local device | m10i |
| | <re-master/> | Empty element included if the script is executed on the master Routing Engine | |
| | <routing-engine-name> | Routing Engine on which the script is executed | re0 |
| | <tty> | TTY of the user's session | /dev/ttyp1 |
| | <script-type> | Type of script being executed | op |
| <junos-context><br><user-context> | <class-name> | Login class of the user executing the script | superuser |
| | <login-name> | Login name of the user executing the script. For AAA access, this is the RADIUS/TACACS username. | jsmith |
| | <uid> | User ID number of the user executing the script as defined in the device configuration | 2999 |
| | <user> | Local name of the user executing the script. Junos OS uses the local name for authentication. It might differ from the **login-name** used for AAA authentication. | root |
| <junos-context><br><op-context><br><br>(op scripts only) | <via-url> | Empty element included if the remote op script is executed using the **op url** command | |

Table 24: Global Variable $junos-context Available to Automation Scripts *(continued)*

| Parent Node | Node | Description | Example content |
|---|---|---|---|
| <junos-context><br><commit-context><br><br>(commit scripts only) | <commit-boot/> | Empty element included when the commit occurs at boot time | |
| | <commit-check/> | Empty element included when a **commit check** is performed | |
| | <commit-comment> | User comment regarding the commit | Commit to fix forwarding issue |
| | <commit-confirm/> | Empty element included when a **commit confirmed** is performed | |
| | <commit-sync/> | Empty element included when a **commit synchronize** is performed | |
| | <database-path/> | Element specifying the location of the session's pre-inheritance candidate configuration. For normal configuration sessions, the value of the element is the location of the normal candidate database. For private configuration sessions, the value of the element is the location of the private candidate database. When the **<get-configuration> database-path** attribute is set to this value, the commit script retrieves the corresponding pre-inheritance candidate configuration. | |

The **$junos-context** variable is a node-set. Therefore, you can access the child elements throughout a script by including the proper XPath expression. The following example commit script writes a message to the system log file if the commit is performed during initial boot-up. The message is given a facility value of **daemon** and a severity value of **info**. For more information, see **syslog()**.

```
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 ($junos-context/commit-context/commit-boot) {
    expr jcs:syslog("daemon.info", "This is boot-time commit");
  }
  else {
     /* Do this ... */
  }
}
```

**Related Documentation**

- Junos Script Automation: Understanding Extension Functions in the jcs and slax Namespaces on page 95

- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- SLAX Parameters Overview on page 49
- SLAX Variables Overview on page 53
- XSLT Parameters Overview on page 26
- XSLT Variables Overview on page 29

CHAPTER 8

# Summary of XPath and XSLT Constructs

This chapter discusses the following topics:



## Summary of Standard XPath and XSLT Functions Referenced in This Guide

Junos OS commit scripts, op scripts, and event scripts support all functions defined in the Extensible Markup Language Path Language (XPath) and Extensible Stylesheet Language Transformations (XSLT) scripting languages. XPath and XSLT share the same function library. This section describes only the functions referenced in this guide. The functions are organized alphabetically.



### concat()

**Syntax** *string* concat(*string*, *string*+)

**Description** Return the concatenation of the arguments.

**Usage Examples** See "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Controlling IS-IS and MPLS Interfaces" on page 379, "Example: Adding T1 Interfaces to a RIP Group"

on page 348, "Example: Configuring Administrative Groups for LSPs" on page 362, and "Example: Configuring Dual Routing Engines" on page 370.

**Related Documentation**

- contains() on page 144
- starts-with() on page 146
- string-length() on page 146
- substring-after() on page 147
- substring-before() on page 147

## contains()

**Syntax** *boolean* contains(*string*, *string*)

**Description** Return TRUE if the first string argument contains the second string argument, otherwise return FALSE.

**Usage Examples** See "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- concat() on page 143
- starts-with() on page 146
- string-length() on page 146
- substring-after() on page 147
- substring-before() on page 147

## count()

**Syntax** *number* count(*node-set*)

**Description** Return the number of nodes in the argument node-set.

**Usage Examples** See "Example: Limiting the Number of E1 Interfaces" on page 401.

**Related Documentation**

- last() on page 144
- name() on page 145
- not() on page 145
- position() on page 146

## last()

**Syntax** *number* last()

**Description** Return the index of the last node in the list that is currently being evaluated, which is equal to the number of items in the processed node list.

**Usage Examples** See "Example: Limiting the Number of E1 Interfaces" on page 401.

**Related Documentation**
- count() on page 144
- name() on page 145
- not() on page 145
- position() on page 146

## name()

**Syntax** *string* name(*<node-set>*)

**Description** Return the full name of the first node in the node set, including the prefix for its namespace declared in the source document. If no argument is passed, the function returns the full name of the context node.

**Usage Examples** See jcs:emit-change Template.

**Related Documentation**
- count() on page 144
- last() on page 144
- not() on page 145
- position() on page 146

## not()

**Syntax** *boolean* not(*boolean*)

**Description** Return TRUE if the argument is FALSE, and FALSE if the argument is TRUE.

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435, "Example: Controlling IS-IS and MPLS Interfaces" on page 379, "Example: Configuring a Default Encapsulation Type" on page 367, "Example: Controlling LDP Configuration" on page 383, "Example: Adding a Final then accept Term to a Firewall" on page 344, "Example: Configuring Administrative Groups for LSPs" on page 362, "Example: Configuring Dual Routing Engines" on page 370, and "Example: Preventing Import of the Full Routing Table" on page 429.

**Related Documentation**
- count() on page 144
- last() on page 144
- name() on page 145

- position() on page 146

## position()

**Syntax** *number* position()

**Description** Return the position of the context node among the list of nodes that are currently being evaluated.

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344 and "Example: Prepending a Global Policy" on page 425.

**Related Documentation**
- count() on page 144
- last() on page 144
- name() on page 145
- not() on page 145

## starts-with()

**Syntax** *boolean* starts-with(*string*, *string*)

**Description** Return TRUE if the first string argument starts with the second string argument, otherwise return FALSE.

**Usage Examples** See "Example: Imposing a Minimum MTU Setting" on page 395, "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Limiting the Number of ATM Virtual Circuits" on page 397, "Example: Adding T1 Interfaces to a RIP Group" on page 348, "Example: Configuring a Default Encapsulation Type" on page 367, and "Example: Configuring Dual Routing Engines" on page 370.

**Related Documentation**
- concat() on page 143
- contains() on page 144
- string-length() on page 146
- substring-after() on page 147
- string-length() on page 146
- substring-after() on page 147
- substring-before() on page 147

## string-length()

**Syntax** *number* string-length(<*string*>)

**Description** Return the number of characters in the string. If the argument is omitted, it returns the string value of the context node.

**Usage Examples** See "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**
- concat( ) on page 143
- contains( ) on page 144
- starts-with( ) on page 146
- substring-after( ) on page 147
- substring-before( ) on page 147

## substring-after( )

**Syntax** *string* substring-after(*string*, *string*)

**Description** Return the portion of the first string argument that follows the occurrence of the second argument substring within the first. If the second string is not contained in the first string, or if the second string is empty, the function returns an empty string.

**Usage Examples** See "Example: Limiting the Number of E1 Interfaces" on page 401 and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**
- concat( ) on page 143
- contains( ) on page 144
- starts-with( ) on page 146
- string-length( ) on page 146
- substring-before( ) on page 147

## substring-before( )

**Syntax** *string* substring-before(*string*, *string*)

**Description** Return the portion of the first string argument that precedes the occurrence of the second argument substring within the first. If the second string is not contained in the first string, or if the second string is empty, the function returns an empty string.

**Usage Examples** See "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**
- concat( ) on page 143
- contains( ) on page 144
- starts-with( ) on page 146
- string-length( ) on page 146

- substring-after( ) on page 147

## Summary of Standard XSLT Elements and Attributes Referenced in This Guide

Junos OS commit scripts, op scripts, and event scripts support all elements and attributes defined in the Extensible Markup Language Path Language (XPath) and Extensible Stylesheet Language Transformations (XSLT) scripting languages. This section describes only the elements and attributes referenced in this guide. The elements and attributes are organized alphabetically.

- xsl:apply-templates on page 149
- xsl:call-template on page 149
- xsl:choose on page 150
- xsl:comment on page 151
- xsl:copy-of on page 151
- xsl:element on page 152
- xsl:for-each on page 152
- xsl:if on page 153
- xsl:import on page 153
- xsl:otherwise on page 154
- xsl:param on page 155
- xsl:stylesheet on page 156
- xsl:template on page 157
- xsl:text on page 159
- xsl:value-of on page 159
- xsl:variable on page 160
- xsl:when on page 161
- xsl:with-param on page 161

## xsl:apply-templates

**Syntax**

```
<xsl:apply-templates select="node-set-expression">
  <xsl:with-param name="qualified-name" select="expression">
    ...
  </xsl:with-param>
</xsl:apply-templates>
```

**Description** Apply one or more templates, according to the value of the **select** attribute. If the **select** attribute is not included, the script recursively processes all child nodes of the current node. If the **select** attribute is present, the processor only applies templates to the child elements that match the expression of the **select** attribute, which must evaluate to a node-set. The **<xsl:template>** instruction dictates which elements are transformed according to which template. The templates that are applied are passed the parameters specified by the **<xsl:with-param>** elements within the **<xsl:apply-templates>** instruction.

**Attributes** select—(Optional) Selects the nodes to which the processor applies templates. By default, the processor applies templates to the child nodes of the current node.

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344 and "Example: Preventing Import of the Full Routing Table" on page 429.

**Related Documentation**

- XSLT Templates Overview on page 24
- xsl:call-template on page 149
- xsl:param on page 155
- xsl:template on page 157
- xsl:variable on page 160
- xsl:with-param on page 161

## xsl:call-template

**Syntax**

```
<xsl:call-template name="qualified-name">
  <xsl:with-param name="qualified-name" select="expression">
    ...
  </xsl:with-param>
</xsl:call-template>
```

**Description** Call a named template. The **<xsl:with-param>** elements within the **<xsl:call-template>** instruction define the parameters that are passed to the template.

**Attributes** name—Specifies the name of the template to call.

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435, "Example: Imposing a Minimum MTU Setting" on page 395, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- XSLT Templates Overview on page 24
- xsl:apply-templates on page 149
- xsl:param on page 155
- xsl:template on page 157
- xsl:variable on page 160
- xsl:with-param on page 161

## xsl:choose

**Syntax**

```
<xsl:choose>
  <xsl:when test="boolean-expression">
  ...
  </xsl:when>
  <xsl:otherwise>
  ...
  </xsl:otherwise>
</xsl:call-template>
```

**Description** Evaluate multiple conditional tests, and execute instructions for the first test that evaluates to TRUE or execute an optional default set of instructions if all tests evaluate to FALSE. The **<xsl:choose>** instruction contains one or more **<xsl:when>** elements, each of which tests a Boolean expression. If the test evaluates to TRUE, the XSLT processor executes the instructions in the **<xsl:when>** element, and ignores all subsequent **<xsl:when>** elements. The XSLT processor processes only the instructions contained in the first **<xsl:when>** element whose **test** attribute evaluates to TRUE. If none of the **<xsl:when>** elements' **test** attributes evaluate to TRUE, the content of the optional **<xsl:otherwise>** element, if one is present, is processed.

**Usage Examples** See "Example: Configuring Dual Routing Engines" on page 370, "Example: Preventing Import of the Full Routing Table" on page 429, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- XSLT Programming Instructions Overview on page 30
- xsl:for-each on page 152
- xsl:if on page 153
- xsl:otherwise on page 154
- xsl:when on page 161

## xsl:comment

**Syntax**

```
<xsl:comment>
  ...
</xsl:comment>
```

**Description** Generate a comment node within the final document. The content within the **<xsl:comment>** element determines the value of the comment. The content must not contain two hyphens next to each other (- -); this sequence is not allowed in comments.

XSLT files can contain ordinary comments delimited by **<!--** and **-->** such as **<!-- ... Insert your comment here ... -->**, but these are ignored by the processor. To generate a comment within the final document, use an **<xsl:comment>** element.

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344.

**Related Documentation**

- xsl:import on page 153
- xsl:stylesheet on page 156

## xsl:copy-of

**Syntax**

```
<xsl:copy-of select="expression"/>
```

**Description** Create a copy of what is selected by the expression defined in the **select** attribute. Namespace nodes, child nodes, and attributes of the current node are automatically copied as well.

**Attributes** select—XPath expression specifying which nodes to copy.

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435.

**Related Documentation**

- xsl:element on page 152
- xsl:text on page 159
- xsl:value-of on page 159

## xsl:element

**Syntax**

```
<xsl:element name="expression"/>
```

**Description** Create an element node in the output document.

**Attributes** name—Specifies the name of the element to be created. The value of the **name** attribute can be set to an expression that is extracted from the input XML document and evaluated at run time. To do this, enclose an XML element in curly brackets, as in **<xsl:element name="{$isis-level-1}"**.

**Usage Examples** See "Example: Creating a Complex Configuration Based on a Simple Interface Configuration" on page 388.

**Related Documentation**

- xsl:copy-of on page 151
- xsl:text on page 159
- xsl:value-of on page 159

## xsl:for-each

**Syntax**

```
<xsl:for-each select="node-set-expression">
  ...
</xsl:for-each>
```

**Description** Include a looping mechanism that repeats XSL processing for each XML element in the specified node-set. The element nodes are selected by the XPath expression defined by the **select** attribute. Each of the nodes is then processed according to the instructions contained in the **<xsl:for-each>** element.

**Attributes** select—Specifies an XPath expression that selects the nodes to be processed.

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435, "Example: Imposing a Minimum MTU Setting" on page 395, "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Adding T1 Interfaces to a RIP Group" on page 348, "Example: Configuring Administrative Groups for LSPs" on page 362, and "Example: Configuring Dual Routing Engines" on page 370.

**Related Documentation**

- XSLT Programming Instructions Overview on page 30
- XPath Overview on page 22
- xsl:choose on page 150
- xsl:if on page 153
- xsl:otherwise on page 154
- xsl:when on page 161

## xsl:if

**Syntax**

```
<xsl:if test="expression">
  ...
</xsl:if>
```

**Description** Include a conditional construct that causes instructions to be processed if the expression held in the **test** attribute evaluates to **TRUE**.

**Attributes** test—Specifies the expression to evaluate.

**Usage Examples**

- Example: Adding T1 Interfaces to a RIP Group on page 348
- Example: Configuring Dual Routing Engines on page 370
- Example: Limiting the Number of E1 Interfaces on page 401
- Example: Requiring and Restricting Configuration Statements on page 435

**Related Documentation**

- XSLT Programming Instructions Overview on page 30
- **xsl:choose on page 150**
- **xsl:for-each on page 152**
- **xsl:otherwise on page 154**
- **xsl:when on page 161**

## xsl:import

**Syntax**

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

**Description** Import rules from an external style sheet. Provides access to all the declarations and templates within the imported style sheet, and allows you to override them with your own if needed. Any **<xsl:import>** elements must be the first elements within the style sheet, the first children of the **<xsl:stylesheet>** document element. The path can be any URI. The **../import/junos.xsl** path shown in the syntax is standard for all commit scripts, op scripts, and event scripts.

Imported rules are overwritten by any subsequent matching rules within the importing style sheet. If more than one style sheet is imported, the style sheets imported last override each previous import where the rules match.

**Attributes** href—Specifies the location of the imported style sheet.

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344, "Example: Configuring a Default Encapsulation Type" on page 367, "Example: Configuring Dual Routing Engines" on page 370, "Example: Controlling IS-IS and MPLS Interfaces" on page 379, "Example: Prepending a Global Policy" on page 425, and "Example: Preventing Import of the Full Routing Table" on page 429.

**Related Documentation**

- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137
- Required Boilerplate for Commit Scripts on page 270
- Required Boilerplate for Event Scripts on page 665
- Required Boilerplate for Op Scripts on page 463
- **xsl:stylesheet on page 156**

## xsl:otherwise

**Syntax**

```
<xsl:otherwise>
  ...
</xsl:otherwise>
```

**Description** Within an **<xsl:choose>** instruction, include a default set of instructions that are processed if none of the expressions defined in the **test** attributes of the **<xsl:when>** elements evaluate to **TRUE**.

**Usage Examples** See "Example: Configuring Dual Routing Engines" on page 370 and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- XSLT Programming Instructions Overview on page 30
- **xsl:choose on page 150**
- **xsl:for-each on page 152**
- **xsl:if on page 153**
- **xsl:when on page 161**

## xsl:param

**Syntax**

```
<xsl:param name="qualified-name" select="expression">
  ...
</xsl:param>
```

**Description** Declare a parameter for a template or for the style sheet as a whole. A template parameter must be declared within the template element. A global parameter, the scope of which is the entire style sheet, must be declared at the top level of the style sheet.

**Attributes** name—Defines the name of the parameter.

select—(Optional) XPath expression defining the default value for the parameter, which is used if the person or client application that executes the script does not explicitly provide a value. The **select** attribute or the content of the **<xsl:param>** element can define the default value. Do not specify both a **select** attribute and content; we recommend using the **select** attribute so as not to create a result tree fragment.

**Usage Examples**

- Example: Imposing a Minimum MTU Setting on page 395
- Example: Limiting the Number of ATM Virtual Circuits on page 397
- Example: Limiting the Number of E1 Interfaces on page 401
- Example: Preventing Import of the Full Routing Table on page 429
- Example: Requiring and Restricting Configuration Statements on page 435

**Related Documentation**

- XSLT Parameters Overview on page 26
- XSLT Templates Overview on page 24
- **xsl:apply-templates on page 149**
- **xsl:call-template on page 149**
- **xsl:template on page 157**
- **xsl:variable on page 160**
- **xsl:with-param on page 161**

## xsl:stylesheet

**Syntax**

```
<xsl:stylesheet version="1.0" xmlns:ext="URI">
  <xsl:import href="../import/junos.xsl"/>
  ...
</xsl:stylesheet>
```

**Description**

Include the document element for the style sheet. This element defines the root element of the style sheet, which contains all the top-level elements such as global variable and parameter declarations, import elements, and templates. Optionally, namespace mappings, which include an extension prefix and Uniform Resource Identifier (URI), can be included as attributes in the opening **<xsl:stylesheet>** tag.

Any **<xsl:import>** elements must be the first elements within the style sheet, the first children of the **<xsl:stylesheet>** document element. The path can be any Uniform Resource Identifier (URI). The **../import/junos.xsl** path shown in the syntax is standard for all commit scripts, op scripts, and event scripts.

**Attributes**

version—Specifies the version of XSLT that is being used. Junos OS supports XSLT version 1.0.

xmlns:*ext*="*URI*—(Optional) Maps a namespace prefix to the URI for extension elements.

**Usage Examples**

- Example: Adding a Final then accept Term to a Firewall on page 344
- Example: Configuring Administrative Groups for LSPs on page 362
- Example: Configuring a Default Encapsulation Type on page 367
- Example: Customizing Output of the show interfaces terse Command Using an Op Script on page 497

**Related Documentation**

- Required Boilerplate for Commit Scripts on page 270
- Required Boilerplate for Event Scripts on page 665
- Required Boilerplate for Op Scripts on page 463
- XSLT Namespace on page 21
- **xsl:import on page 153**

## xsl:template

**Syntax**

```
<xsl:template match="pattern" mode="qualified-name" name="qualified-name
priority="integer">
  <xsl:param name="qualified-name" select="expression">
    ...
  </xsl:param>
  ...
</xsl:stylesheet>
```

**Description** Declare a template that contains rules to apply when a specified node is matched. The **match** attribute associates the template with an XML element. The **match** attribute can also be used to define a template for a whole branch of an XML document. For example, **match="/"** matches the root element of the document. Although the **match** and **name** attributes are optional, one of the two attributes must be included in the template definition.

When templates are applied to a node set using the **<xsl:apply-templates>** instruction, they might be applied in a particular mode; the **mode** attribute in the **<xsl:template>** instruction indicates the mode in which a template needs to be applied for the template to be used. If templates are applied in the specified mode, the **match** attribute is used to determine whether the template can be used with the particular node. If more than one template matches a node in the specified mode, the priority attribute determines which template is used. The highest priority wins. If no priority is specified explicitly, the priority of a template is determined by the **match** attribute.

You can pass template parameters using the **<xsl:with-param>** element. To receive a parameter, the template must contain an **<xsl:param>** element that declares a parameter of that name. These parameters are listed before the body of the template, which is used to process the node and create a result.

**Attributes**

match—(Optional) XPath expression specifying the nodes to which to apply the template. If this attribute is omitted, the **name** attribute must be included.

mode—(Optional) Indicate the mode in which a template needs to be applied for the template to be used.

name—(Optional) Specify a name for the template. Named templates can be explicitly called with the **<xsl:call-template>** element. If the **name** attribute is omitted, the **match** attribute must be included.

priority—(Optional) Specify a numeric priority for the template.

**Usage Examples**

- Example: Adding a Final then accept Term to a Firewall on page 344
- Example: Adding T1 Interfaces to a RIP Group on page 348
- Example: Automatically Configuring Logical Interfaces and IP Addresses on page 355
- Example: Customizing Output of the show interfaces terse Command Using an Op Script on page 497
- Example: Requiring and Restricting Configuration Statements on page 435

**Related Documentation**

- XSLT Templates Overview on page 24
- XSLT Parameters Overview on page 26
- xsl:apply-templates on page 149
- xsl:call-template on page 149
- xsl:param on page 155
- xsl:variable on page 160
- xsl:with-param on page 161

## xsl:text

**Syntax**

```
<xsl:text>
  ...
</xsl:text>
```

**Description** Insert literal text in the output.

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435, "Example: Imposing a Minimum MTU Setting" on page 395, "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Controlling IS-IS and MPLS Interfaces" on page 379, and "Example: Adding a Final then accept Term to a Firewall" on page 344.

## xsl:value-of

**Syntax**

```
<xsl:value-of select="expression"/>
```

**Description** Extract the value of an XML element and insert it into the output. The **select** attribute specifies the XPath expression that is evaluated. In the XPath expression, use @ to access attributes of elements. Use " **.** " to access the contents of the element itself. If the result is a node set, the **<xsl:value-of>** instruction adds the string value of the first node in that node set; none of the structure of the node is preserved. To preserve the structure of the node, you must use the **<xsl:copy-of>** instruction instead.

**Attributes** select—XPath expression specifying the node or attribute to evaluate.

**Usage Examples**

- Example: Automatically Configuring Logical Interfaces and IP Addresses on page 355
- Example: Configuring Administrative Groups for LSPs on page 362
- Example: Controlling IS-IS and MPLS Interfaces on page 379
- Example: Imposing a Minimum MTU Setting on page 395
- Example: Limiting the Number of E1 Interfaces on page 401

**Related Documentation**

- xsl:copy-of on page 151

## xsl:variable

**Syntax**

```
<xsl:variable name="qualified-name" select="expression">
  ...
</xsl:variable>
```

**Description** Declare a local or global variable. If the **<xsl:variable>** instruction appears at the top level of the style sheet as a child of the **<xsl:stylesheet>** document element, it is a global variable with a scope that includes the entire style sheet. Otherwise, it is a local variable with a scope of its following siblings and their descendants.

**Attributes**

name—Specifies the name of the variable. After declaration, the variable can be referred to within XPath expressions using this name, prefixed with the **$** character.

select—(Optional) Determines the value of the variable. The value of the variable is determined either by the **select** attribute or by the contents of the **<xsl:variable>** element. Do not specify both a **select** attribute and some content; we recommend using the **select** attribute so as not to create a result tree fragment.

**Usage Examples** See "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Limiting the Number of ATM Virtual Circuits" on page 397, "Example: Configuring Administrative Groups for LSPs" on page 362, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- XSLT Variables Overview on page 29
- xsl:apply-templates on page 149
- xsl:call-template on page 149
- xsl:param on page 155
- xsl:template on page 157
- xsl:with-param on page 161

## xsl:when

**Syntax**

```
<xsl:when test="boolean-expression">
  ...
</xsl:when>
```

**Description** Within an **<xsl:choose>** instruction, specify a set of processing instructions that are executed when the expression specified in the **test** attribute evaluates to **TRUE**. The XSLT processor processes only the instructions contained in the first **<xsl:when>** element whose **test** attribute evaluates to **TRUE**. If none of the **<xsl:when>** elements' **test** attributes evaluate to **TRUE**, the content of the **<xsl:otherwise>** element, if there is one, is processed.

**Attributes** test—Specifies a Boolean expression.

**Usage Examples**

- Example: Automatically Configuring Logical Interfaces and IP Addresses on page 355
- Example: Configuring Dual Routing Engines on page 370
- Example: Preventing Import of the Full Routing Table on page 429

**Related Documentation**

- XSLT Programming Instructions Overview on page 30
- xsl:choose on page 150
- xsl:for-each on page 152
- xsl:if on page 153
- xsl:otherwise on page 154

## xsl:with-param

**Syntax**

```
<xsl:with-param name="qualified-name" select="expression">
  ...
</xsl:with-param>
```

**Description** Specify a parameter to pass into a template. This element can be used when applying templates with the **<xsl:apply-templates>** instruction or when calling templates with the **<xsl:call-template>** instruction.

**Attributes** name—Specifies the name of the parameter.

select—(Optional) XPath expression specifying the value of the parameter. The value of the parameter is determined either by the **select** attribute or by the contents of the **<xsl:with-param>** element. Do not specify both a **select** attribute and content. We recommend using the **select** attribute to set the parameter so as to prevent the parameter from being passed a result tree fragment as its value.

**Usage Examples** See "Example: Configuring Dual Routing Engines" on page 370, "Example: Preventing Import of the Full Routing Table" on page 429, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- XSLT Templates Overview on page 24
- xsl:apply-templates on page 149
- xsl:call-template on page 149
- xsl:param on page 155
- xsl:template on page 157
- xsl:variable on page 160

CHAPTER 9

# Summary of SLAX Statements

This chapter summarizes the Stylesheet Language Alternative Syntax (SLAX) statements, with brief examples and Extensible Stylesheet Language Transformations (XSLT) equivalents. The statements are organized alphabetically.

## append

**Syntax** append *name* += *value*;

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Append a value to the node set contained in a mutable variable. The variable must be defined using the **mvar** statement.

**Attributes** *name*—Name of the mutable variable.

*value*—Value to append to the node set.

**SLAX Example** The following snippet appends the **<item>** element and **<name>** and **<size>** child elements to the node set contained in the mutable variable **block**:

```
mvar $block;
set $block = <block> "item list";

for $item (list) {
  append $block += <item> {
    <name> $item/name;
    <size> $item/size;
  }
}
```

**Related Documentation**

- SLAX Variables Overview on page 53
- mvar on page 187
- set on page 200

## apply-imports

**Syntax**

```
apply-imports;
```

**Release Information**

Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Apply a template rule from an external file or style sheet. By default, template rules in the main script have precedence over equivalent imported template rules. Use this statement to process the context node using the imported match template rule from the external source.

The **apply-imports** statement mimics the **<xsl:apply-imports>** element, allowing the script to invoke any imported templates.

**SLAX Example**

In the example, the main script imports the file **route-rules.slax**. The **apply-imports** statement invokes the imported template rule for **<route>** elements.

```
version 1.1;
import "route-rules.slax";

match route {
  <routes> {
    apply-imports;
  }
}
```

The imported file contains a template rule for **<route>** elements.

```
/* route-rules.slax */
version 1.1;

match route {
  <new> {
    apply-templates *[@changed];
  }
}
```

**Related Documentation**

- import on page 182

## apply-templates

**Syntax**

```
apply-templates expression;
```

**Release Information**

Statement introduced in version 1.0 of the SLAX language.

**Description** Apply one or more templates, according to the value of the node-set expression. If a node-set expression is not specified, the script recursively processes all child nodes of the current node. If a node-set expression is specified, the processor only applies templates to the child elements that match the node-set expression. The **template** statement dictates which elements are transformed according to which template. The templates that are applied are passed the parameters specified by the **with** statement within the **apply-templates** statement block.

**Attributes** *expression*—(Optional) Selects the nodes to which the processor applies templates. By default, the processor applies templates to the child nodes of the current node.

**SLAX Example**

```
match configuration {
  apply-templates system/host-name;
}
```

**XSLT Equivalent**

```
<xsl:template match="configuration">
  <xsl:apply-templates select="system/host-name"/>
</xsl:template>
```

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344 and "Example: Preventing Import of the Full Routing Table" on page 429.

**Related Documentation**

- SLAX Templates Overview on page 44
- call on page 169
- match on page 185
- mode on page 186
- priority on page 196
- template on page 202
- with on page 209

## attribute

**Syntax**

```
attribute attribute-name {
  attribute-value;
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Create an attribute with the given name. The attribute value is defined by a block of statements, which must be placed inside a set of braces.

**Attributes** *attribute-name*—Name of the attribute, which can be an XPath expression or a string. Enclose string arguments in quotes.

*attribute-value*—A block of statements enclosed in curly braces that defines the attribute value.

**SLAX Example**

In the following example, the **<book>** element is output to the result tree with an attribute named **format**, which has the value "PDF":

```
<book> {
  attribute "format" {
    expr "PDF";
  }
}
```

In the following example, the value of the **<name>** node (rather than the literal string "name") is used to create an XML attribute with a value of "from-" concatenated with the contents of the address node. Node values are selected from the current context.

```
<source> {
  attribute name {
    expr "from-" _ address;
  }
}
```

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42
- attribute-set on page 167
- element on page 173
- use-attribute-sets on page 206

## attribute-set

**Syntax**

```
attribute-set attribute-set-name {;
  attribute attribute-name1 { attribute-value1; }
  attribute attribute-name2 { attribute-value2; }
  use-attribute-sets attribute-set-name2;
...
}
```

**Release Information**

Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Define a collection of attributes that can be used repeatedly. The **attribute-set** statement must be defined as a top-level statement in the script. The attribute set name is a string argument. The attribute set contents define the attributes to include in the collection. The contents can include individual **attribute** statements, which define attributes as a name and value pair, and they can include **use-attribute-sets** statements, which add the attributes from a previously defined attribute set to the current set.

To apply the attributes in an attribute set to a specific element, include the **use-attribute-sets** statement under that element and reference the attribute set name.

**Attributes**

*attribute-set-name*—Name of the attribute set, which must be a string. To add the attribute set to an element, reference this name in the **use-attribute-sets** statement.

*attribute-name*—Name of the individual attribute to add to the set.

*attribute-value*—A block of statements enclosed in curly braces that defines the attribute value.

**SLAX Example**

The following example creates two attribute sets: **table-attributes** and **table-attributes-ext**. The **table-attributes-ext** set includes all of the attributes that are already defined in the **table-attributes** set through use of the **use-attributes-sets** statement. In the main script body, the **table-attributes-ext** attribute set is applied to the **<table>** element. The **<table>** element includes the four attributes: **order**, **cellpadding**, **cellspacing**, and **border**.

```
version 1.1;

var $cellpadding = "0";
var $cellspacing = "10";

attribute-set table-attributes {
  attribute "order" { expr "0"; }
  attribute "cellpadding" { expr $cellpadding; }
  attribute "cellspacing" { expr $cellspacing; }
}
attribute-set table-attributes-ext {
  use-attribute-sets table-attributes;
  attribute "border" { expr "0"; }
}

match / {
  ...
  <table> {
    use-attribute-sets table-attributes-ext;
  }
}
```

**XSLT Equivalent**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:variable name="cellpadding" select="0"/>
  <xsl:variable name="cellspacing" select="10"/>
  <xsl:attribute-set name="table-attributes">
    <xsl:attribute name="order">
      <xsl:text>0</xsl:text>
    </xsl:attribute>
    <xsl:attribute name="cellpadding">
      <xsl:value-of select="$cellpadding"/>
    </xsl:attribute>
    <xsl:attribute name="cellspacing">
      <xsl:value-of select="$cellspacing"/>
    </xsl:attribute>
  </xsl:attribute-set>
 <xsl:attribute-set name="table-attributes-ext" use-attribute-sets="table-attributes">

    <xsl:attribute name="border">
      <xsl:text>0</xsl:text>
    </xsl:attribute>
  </xsl:attribute-set>
  <xsl:template match="/">
```

```
    <table use-attribute-sets="table-attributes-ext"/>
  </xsl:template>
</xsl:stylesheet>
```

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42
- attribute on page 166
- element on page 173
- use-attribute-sets on page 206

## call

**Syntax**

```
call template-name (parameter-name = value) {
  /* code */
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Call a named template. You can pass parameters into the template by including a comma-separated list of parameters, with the parameter name and an optional equal sign (=) and value expression. If a value is not specified, the current value of the parameter is passed to the template.

You can declare additional parameters inside the code block using the **with** statement.

**Attributes** *template-name*—Specifies the name of the template to call.

**SLAX Example**

```
match configuration {
  var $name-servers = name-servers/name;
  call temp();
  call temp($name-servers, $size = count($name-servers));
  call temp() {
    with $name-servers;
    with $size = count($name-servers);
  }

  template temp($name-servers, $size = 0) {
    <output> "template called with size " _ $size;
  }
}
```

**XSLT Equivalent**

```
<xsl:template match="configuration">
  <xsl:variable name="name-servers" select="name-servers/name"/>
  <xsl:call-template name="temp"/>
  <xsl:call-template name="temp">
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
  <xsl:call-template name="temp">
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
```

```
</xsl:template>

<xsl:template name="temp">
  <xsl:param name="name-servers"/>
  <xsl:param name="size" select="0"/>
  <output>
    <xsl:value-of select="concat('template called with size ', $size)"/>
  </output>
</xsl:template>
```

**Usage Examples** See "Example: Requiring and Restricting Configuration Statements" on page 435, "Example: Imposing a Minimum MTU Setting" on page 395, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Templates Overview on page 44
- apply-templates on page 165
- match on page 185
- mode on page 186
- priority on page 196
- template on page 202
- with on page 209

## copy-node

**Syntax**

```
copy-node;
```

```
copy-node {
  /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Copy the current node including namespace nodes to the result tree, but do not copy any attribute or child nodes. The optional body is a block of statements that emit additional nodes inside that copy.

**SLAX Example**

```
copy-node {
  <that> "one";
}
```

**XSLT Equivalent**

```
<xsl:copy>
  <that>
    <xsl:value-of select="one"/>
  </that>
</xsl:copy>
```

Related Documentation
- copy-of on page 171
- xsl:copy-of on page 151

## copy-of

**Syntax**

```
copy-of expression;
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Copy the specified node including namespace nodes, child nodes, and attributes of that node. The argument is an XPath expression that specifies which nodes to copy.

**Attributes** *expression*—XPath expression that specifies which nodes to copy.

**SLAX Example**

```
copy-of configuration/protocols/bgp;
```

**XSLT Equivalent**

```
<xsl:copy-of select="configuration/protocols/bgp"/>
```

Related Documentation
- copy-node on page 170
- xsl:copy-of on page 151

## decimal-format

**Syntax**

```
decimal-format format-name {
  decimal-separator character;
  digit character ;
  grouping-separator character;
  infinity string;
  minus-sign character;
  nan string;
  pattern-separator character;
  percent character;
  per-mille character;
  zero-digit character;
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Define formatting parameters for use by the **format-number()** XPath function. The **decimal-format** statement must be defined as a top-level statement in the script.

**Attributes**

decimal-format *format-name*—Decimal-format identifier, which is passed as the third argument to the **format-number()** XPath function.

decimal-separator *character*—Character used as the decimal sign. The default is the period (.).

digit *character*—Character used to represent a digit in a pattern. The default is the number sign (#).

grouping-separator *character*—Character used as the digit group separator or the thousands separator. The default is the comma (,).

infinity *string*—String used to represent infinity. The default is "Infinity".

minus-sign *character*—Character used as the minus sign. The default is the hyphen (-).

nan *string*—String used to represent NaN. The default is "NaN".

pattern-separator *character*—Character used to separate patterns. The first pattern is used for positive numbers, and the second pattern is used for negative numbers. The default is the semicolon (;).

percent *character*—Character used as the percent sign. The default is the percent character (%).

per-mille *character*—Character used as a per mille sign. The default is the Unicode per mille sign (\x2030 or ‰).

zero-digit *character*—Character used as zero. The default is the number zero (0).

**SLAX Example**

The following code snippet lists the defaults for the decimal-format parameters, and uses the defined decimal format in the **format-number** XPath function:

```
version 1.1;

decimal-format default-format {
  decimal-separator "." ;
  digit "#" ;
  grouping-separator "," ;
  infinity "Infinity" ;
  minus-sign "-" ;
  nan "NaN";
  pattern-separator ";" ;
  percent "%";
  per-mille "\x2030";
  zero-digit "0" ;
}

match / {
...
  var $number = -14560302.5;
   expr format-number($number, "###,###.00", "default-format");
}

/* output is -14,560,302.50 */
```

**XSLT Equivalent**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

 <xsl:decimal-format name="default-format" decimal-separator="." digit="#"
   grouping-separator="," infinity="Infinity" minus-sign="-" NaN="NaN"
   pattern-separator=";" percent="%" per-mille="\x2030" zero-digit="0"/>

 <xsl:template match="/">
```

```
    <xsl:variable name="number" select="-14560302.5"/>
    <xsl:value-of select="format-number($number, '###,###.00', 'default-format')"/>

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

**Related Documentation**

- output-method on page 192

## element

**Syntax**

```
element name {
 /* element contents */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Create an element node with the given name. The element name can be an XPath expression or a string. The element contents must be placed inside curly braces.

**Attributes** *name*—Name of the element, which can be an XPath expression or a string. Enclose string arguments in quotes.

**SLAX Example** The following sample code uses the value of the **name** node (rather than the literal string "name") to create an XML element, whose contents are an empty element with a name of "**from-**" concatenated with the value of the address node. Node values are selected from the current context.

```
for-each (list/item) {
  element name {
    element "from-" _ address;
  }
}
```

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42

## else

**Syntax**

```
if (expression) {
    /* code */
}
else {
  /* code */
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Include a default set of instructions that are processed if the preceding **if** and **else if** statements evaluate to **FALSE**.

**SLAX Example**

```
if (starts-with(name, "fe-")) {
  if (mtu < 1500) {
    /* Select the Fast Ethernet interfaces with low MTUs */
  }
}
else {
  if (mtu > 8096) {
    /* Select the non-Fast Ethernet interfaces with high MTUs */
  }
}
```

**XSLT Equivalent**

```
<xsl:choose>
  <xsl:when select="starts-with(name, 'fe-')">
    <xsl:if test="mtu &lt; 1500">
      <!-- Select with Fast Ethernet interfaces with low MTUs -->
    </xsl:if>
  </xsl:when>
  <xsl:otherwise>
    <xsl:if test="mtu &gt; 8096">
      <!-- Select the non-Fast Ethernet interfaces with high MTUs -->
    </xsl:if>
  </xsl:otherwise>
</xsl:choose>
```

**Usage Examples** See "Example: Configuring Dual Routing Engines" on page 370 and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Statements Overview on page 56
- else if on page 174
- for-each on page 178
- if on page 181

## else if

**Syntax**

```
if (expression) {
    /* code */
}
else if (expression) {
    /* code */
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Include instructions that are processed if the expression defined in the preceding **if** statement evaluates to **FALSE** and the expression defined in the **else if** statement evaluates to **TRUE**. Multiple **else if** statements can be included, but the processor only executes the instructions contained in the first **else if** statement whose expression evaluates to **TRUE**. All subsequent **else if** statements are ignored.

**SLAX Example**

```
var $description2 = {
  if (description) {
```

```
  expr description;
 }
 else if (../description) {
  expr ../description;
 }
 else {
  expr "no description found";
 }
}
```

**XSLT Equivalent**

```
<xsl:variable name="description2">
 <xsl:choose>
  <xsl:when test="description">
    <xsl:value-of select="description"/>
  </xsl:when>
  <xsl:when test="../description">
   <xsl:value-of select="../description"/>
  </xsl:when>
  <xsl:otherwise>unknown</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
```

**Usage Examples**

See "Example: Configuring Dual Routing Engines" on page 370 and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Statements Overview on page 56
- else on page 173
- for-each on page 178
- if on page 181

## expr

**Syntax**

expr *expression*;

**Release Information**

Statement introduced in version 1.0 of the SLAX language.

**Description**

Generate the string value of an XPath expression and add it to the result tree. The XPath expression might consist of a function call, a location path, a literal number, or a string. SLAX-specific operators are permitted. This statement cannot be used at the top-level of a script. It can only appear within a code block. By default, characters such as "<", ">", and "&" are escaped into proper XML as "<", ">", and "&", respectively.

The **expr** statement is most commonly used to invoke functions that return no results, for conditional variable assignment, and to return text content from a template.

**Attributes**

*expression*—XPath expression to evaluate. The resulting string is added to the result tree.

**SLAX Example**

```
expr "Test: ";
expr substring-before(name, ".");
```

```
expr status;
expr jcs:output("Test");
```

**XSLT Equivalent**

```
<xsl:text>Test: </xsl:text>
<xsl:value-of select="substring-before(name, '.')"/>
<xsl:value-of select="status"/>
<xsl:value-of select="jcs:output('Test')"/>
```

**Usage Examples**

- Example: Adding a Final then accept Term to a Firewall on page 344
- Example: Automatically Configuring Logical Interfaces and IP Addresses on page 355

**Related Documentation**

- XPath Expressions Overview for SLAX on page 43
- message on page 186
- terminate on page 204
- trace on page 204
- uexpr on page 205

## fallback

**Syntax**

```
fallback {
  /* body to execute if extension function
     or element is unavailable */
}
```

**Release Information**

Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Specify statements to use when an extension function or element is not available in the current implementation. The **fallback** statement is enclosed within another instruction element to indicate what fallback code should be run if the script processor does not recognize the enclosing instruction element. The script executes the body of the fallback statement to handle this error condition.

A script might utilize this statement when it is run in environments that support different extension elements.

**SLAX Example**

The following example op script declares the namespace binding **test** with a URI of "test". The code attempts to reference the nonexistent extension element **<test:fake>**, which is not supported, and the code instead executes the fallback instructions.

```
version 1.1;
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";
ns test extension = "test";


match / {
  <op-script-results> {
```

```
    /* Fake extension element */
    <test:fake> {
      expr slax:output( "<test:fake> exists!" );
      fallback {
        expr slax:output( "<test:fake> does not exist." );
      }
    }
  }
}
```

**Related Documentation**

- 

## for

**Syntax**

```
for name (expression) {
  /* code */
}

for name (min ... max) {
  /* code */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Iterate through an integer set or a node set without changing the context, and execute a block of statements using each member of the integer or node set as the value of the given variable.

If the argument is an XPath expression, the variable is assigned each member of the node set selected by the expression in sequence. If the argument is an integer set, the iteration operator (...) generates a sequence of nodes with the value of each integer between the left and right operands. If the left operand is greater than the right operand, the numbers are generated in decreasing order. The variable takes on the value of each integer in sequence. For each iteration, the contents are then evaluated, processed according to the instructions contained in the **for** code block.

**Attributes** *expression*—XPath expression that selects the nodes to be processed.

*max*—Integer or variable that defines the end value of the integer sequence. If the end value is less than the start value, the numbers are generated in decreasing order.

*min*—Integer or variable that defines the starting value of the integer sequence. If the start value is greater than the end value, the numbers are generated in decreasing order.

*name*—Identifier of the **for** loop variable, which takes on the values of each member of the integer or node set. This variable can be referenced within the **for** loop code block.

**SLAX Example**

In the following example, the **for** loop iterates over the **interfaces** node. The XPath expression selects each **name** node that is a child of the **interface** node and that has a value beginning with the 'ge-' designator. The selection is assigned to the **$name** variable, which is used within that iteration of the **for** loop code block. The **for** loop outputs a **<name>** element for each selection. The content of each **<name>** element is the interface name currently stored in the **$name** variable for that iteration. The end result is a list of all Gigabit Ethernet interfaces on the device.

```
for $name (interfaces/interface[starts-with(name, 'ge-')]) {
  <name> {
    expr $name;
  }
}
```

In the following example, the **for** loop iterates over the integers 1 through 3, and the variable **$int** assumes each integer value. For each iteration, the code block generates an **<item>** element, which contains the attribute **item-number** with a value equal to the current integer value of **$int**.

```
for $int (1 ... 3) {
  <item> {
    attribute "item-number" {
      expr $int;
    }
  }
}

/* Output: <item item-number="1"/><item item-number="2"/><item item-number="3"/>
 */
```

**Related Documentation**

- SLAX Statements Overview on page 56
- XPath Overview on page 22
- for-each on page 178

## for-each

**Syntax**

```
for-each (expression) {
  /* code */
}

/* Syntax added in version 1.1 of the SLAX language.*/
for-each (min ... max) {
  /* code */
}
```

**Release Information**

Statement introduced in version 1.0 of the SLAX language.
Support for iteration operator (...) added in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Include a looping mechanism that repeats script processing for each XML element in the specified node set or each value in the integer set.

If the argument is an XPath expression, the element nodes are selected by the value of the XPath expression. If the argument is an integer set, the iteration operator (...) generates

a sequence of nodes with the value of each integer between the left and right operands. If the left operand is greater than the right operand, the numbers are generated in decreasing order. For each iteration, the contents are then evaluated, processed according to the instructions contained in the **for-each** code block.

**Attributes**

for-each *expression*—XPath expression that selects the nodes to be processed.

*max*—Integer or variable that defines the end value of the integer sequence. If the end value is less than the start value, the numbers are generated in decreasing order.

*min*—Integer or variable that defines the starting value of the integer sequence. If the start value is greater than the end value, the numbers are generated in decreasing order.

**SLAX Example**

The following code iterates over each **chassis-sub-module** element that has a part-number child element equal to 750-000610. For each match, the script outputs a **<message>** element with the name of the module and the name and description of the submodule.

```
for-each ($inventory/chassis/chassis-module/
      chassis-sub-module[part-number == '750-000610']) {
  <message> "Down rev PIC in " _ ../name _ ", " _ name _ ": " _ description;
}
```

The following code iterates over the integers 1 through 3. For each iteration, the code block generates an **<item>** element, which contains the attribute **item-number** with a value equal to the current integer value of the set.

```
for-each (1 ... 3) {
  <item> {
    attribute "item-number" {
      expr .;
    }
  }
}

/* Output: <item item-number="1"/><item item-number="2"/><item item-number="3"/>
 */
```

**Usage Examples**

- Example: Adding T1 Interfaces to a RIP Group on page 348
- Example: Configuring Administrative Groups for LSPs on page 362
- Example: Configuring Dual Routing Engines on page 370
- Example: Imposing a Minimum MTU Setting on page 395
- Example: Limiting the Number of E1 Interfaces on page 401
- Example: Requiring and Restricting Configuration Statements on page 435

**Related Documentation**

- SLAX Statements Overview on page 56
- XPath Overview on page 22
- for on page 177

- xsl:for-each on page 152


## function

**Syntax**

```
function function-name (argument-list) {
  ...
  result return-value;
}

function function-name ( ) {
  param param-name1;
  param param-name2;
  param param-name3 = default-value;
  ...
  result return-value;
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Define an extension function that can be used in XPath expressions. The **function** statement must be defined as a top-level statement in the script using a qualified name for the function identifier. The argument list is a comma-separated list of parameter names, which are positionally assigned based on the function call. Trailing arguments can have default values. Alternatively, you can define function parameters inside the function block using the **param** statement. The function body is a set of statements, which should include a **result** statement that defines the return value for the function.

If there are fewer arguments in the function invocation than in the definition, the default values are used for any trailing arguments. If there are more arguments in the function invocation than in the definition, the function call generates an error.

**Attributes** *function-name*—Specifies the name of the function as a qualified name.

*argument-list*—Comma-separated list of parameter names, which are positionally assigned based on the function call. Trailing arguments can have default values.

*return-value*—XML element or XPath expression, scalar value, or a set of instructions providing the return value of the function.

**SLAX Example** The following example defines the function **size**, which has three parameters: **width**, **height**, and **scale**. The default value for **scale** is 1. If the function call argument list does not include the **scale** argument, the calculation uses the default value of 1 for that argument. The function's return value is the product of the **width**, **height**, and **scale** variables enclosed in a **<size>** element.

In the main match template, the function call uses width and height data selected from each **graphic/dimension** element in the source XML file. The script evaluates the function, and the **copy-of** statement emits the return value to the result tree as the contents of the **<out>** element.

```
version 1.1;
ns my = "http://www.example.com/myfunctions";
```

```
function my:size ($width, $height, $scale = 1) {
  result <size> {
    expr $width * $height * $scale;
  }
}

match / {
  for-each (graphic/dimension) {
    <out> {
      copy-of my:size((width/.), (height/.));
    }
  }
}
```

**Related Documentation**

- SLAX Functions Overview on page 47
- param on page 195
- result on page 199

## if

**Syntax**

```
if (expression) {
  /* code */
}
else if (expression) {
    /* code */
}
else {
  /* code */
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Include a conditional construct that causes instructions to be processed if the Boolean expression evaluates to TRUE.

Optionally, you can include multiple **else if** statements following an **if** statement to perform additional conditional tests if the expression in the **if** statement evaluates to **FALSE**. Multiple **else if** statements can be included, but the processor only executes the instructions contained in the first **else if** statement whose expression evaluates to **TRUE**; all subsequent **else if** statements are ignored. The optional **else** statement includes a default set of instructions that are processed if the expressions defined in all associated **if** and **else if** statements evaluate to **FALSE**.

**Attributes** *expression*—Specifies the expression to evaluate.

**SLAX Example**

```
var $description2 = {
  if (description) {
    expr description;
  }
  else if (../description) {
```

```
    expr ../description;
  }
  else {
    expr "no description found";
  }
}
```

**XSLT Equivalent**

```
<xsl:variable name="description2">
  <xsl:choose>
    <xsl:when test="description">
       <xsl:value-of select="description"/>
    </xsl:when>
    <xsl:when test="../description">
      <xsl:value-of select="../description"/>
    </xsl:when>
    <xsl:otherwise>unknown</xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

**Usage Examples** See "Example: Configuring Dual Routing Engines" on page 370, "Example: Preventing Import of the Full Routing Table" on page 429, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Statements Overview on page 56
- else on page 173
- else if on page 174
- for-each on page 178

## import

**Syntax** import *href*;

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Import rules from an external file or style sheet, which provide access to all the declarations and templates within the imported item. Any **import** statements must be the first elements in the script or style sheet. The path can be any URI. The path **../import/junos.xsl** is standard for all commit scripts, op scripts, and event scripts.

Imported rules are overwritten by any subsequent matching rules within the importing script. If more than one file or style sheet is imported, the items imported last override each previous import where the rules match.

**Attributes** *href*—Specifies the location of the imported file or style sheet.

**SLAX Example** In the example, the main script imports the file **route-rules.slax**, which contains a template rule for <route> elements.

```
version 1.1;
import "route-rules.slax";
```

```
match route {
  <routes> {
    apply-imports;
  }
}
```

**Related Documentation**

- Junos Script Automation: Named Templates in the jcs Namespace Overview on page 124
- Junos Script Automation: Global Parameters and Variables in the junos.xsl File on page 137
- Required Boilerplate for Commit Scripts on page 270
- Required Boilerplate for Event Scripts on page 665
- Required Boilerplate for Op Scripts on page 463
- **apply-imports on page 165**

## key

**Syntax**

```
key name {
  match pattern;
  value expression;
}
```

**Release Information**

Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Define a key for use with the **key()** XPath function. Keys are an alternative to IDs and are used to index the nodes within an XML document. The key must be defined as a top-level statement in the script. A **key** definition consists of the key identifier, the nodes to index, and the value that is paired with the key name to reference the matching nodes. The **key()** function is then used to locate the appropriate nodes.

The **key()** function works with the XML document of the current node and uses the specified **key** definition to retrieve nodes that are referenced by a particular name and value. The function arguments are the key name and the desired key's value. The return value is a node set that includes all nodes referenced by that key name and value. If the desired key value is provided as a node set, rather than a string, the returned node set is a union of all the referenced nodes for the key values expressed by the nodes within the node set.

For example, if you define the key:

```
key func {
  match prototype;
  value @name;
}
```

the following code would select **<prototype>** elements that have a **name** attribute with a value of "trace", and then output the value of the child element **<return-type>**:

```
for-each ( key("func", "trace") ) {
  <out> return-type/.;
}
```

**Attributes**

key *name*—Key identifier, which uniquely identifies the key within the script and is passed as the first argument to the **key()** function.

match *pattern*—XPath expression that selects the set of nodes to index.

value *expression*—XPath expression that defines the value of the key.

**SLAX Example**

The following op script creates two **key** definitions, **protocol** and **next-hop**, which are used to retrieve and display all static routes and all routes with a next hop of ge-0/0/0.0 on a device. The script invokes the Junos XML API **get-route-information** command to obtain the route information for the device. The **for-each( $results )** statement changes the current node to the **$results** XML document. The subsequent **for-each** loops use the keys to retrieve all nodes that are indexed according to the key names and values.

The **for-each( key( "protocol", "Static" )** statement uses the **protocol** key definition, which matches on **route-table/rt** elements, to retrieve the desired nodes. The **rt-entry/protocol-name** key value matches the **<protocol-name>** child elements that have the value "Static". The code block executes using **<rt>** as the context node. For each match, the script outputs the value of the **<rt-destination>** element.

The **for-each( key( "next-hop", "ge-0/0/0.0" )** statement uses the "next-hop" key definition, which matches on **route-table/rt** elements, to retrieve the desired nodes. The **rt-entry/nh/via** key value matches the **<via>** child elements that have the value "ge-0/0/0.0". The code block executes using **<rt>** as the context node. For each match, the script outputs the value of the **<rt-destination>** element.

```
version 1.1;

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";

key protocol {
 match route-table/rt;
 value  rt-entry/protocol-name;
}
key next-hop {
 match route-table/rt;
 value  rt-entry/nh/via;
}

match / {
  <op-script-results> {
    var $results = jcs:invoke("get-route-information");

    for-each( $results ) {
      /* Display all static routes */
      <output> "Static routes: ";
```

```
    for-each( key( "protocol", "Static" ) ) {
      <output> rt-destination;
    }
    /* Display all routes with next-hop of ge-0/0/0.0 */
    <output> "Next-hop ge-0/0/0.0: ";
    for-each( key( "next-hop", "ge-0/0/0.0" ) ) {
      <output> rt-destination;
    }
  }
 }
}
```

**Related Documentation**

- XPath Overview on page 22
- XPath Expressions Overview for SLAX on page 43

## match

**Syntax**

```
match expression {
statements;
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Declare a template that contains rules to apply when a specified node is matched. The **match** statement associates the template with an XML element. The **match** statement can also be used to define a template for a whole branch of the XML document For example, **match /** matches the root element of the document.

**Attributes** *expression*—XPath expression specifying the nodes to which to apply the template.

**SLAX Example**

```
match host-name {
  <hello> .;
}
```

**XSLT Equivalent**

```
<xsl:template match="host-name">
  <hello>
    <xsl:value-of select="."/>
  </hello>
</xsl:template>
```

**Usage Examples** "Example: Adding a Final then accept Term to a Firewall" on page 344, "Example: Configuring Dual Routing Engines" on page 370, and "Example: Preventing Import of the Full Routing Table" on page 429.

**Related Documentation**

- apply-templates on page 165
- call on page 169
- mode on page 186
- priority on page 196

- template on page 202
- with on page 209

## message

**Syntax**

message *expression*;

```
message {
 /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Generate an error message that is immediately displayed to the user, typically on the standard error file descriptor. This is different from most script output, which is displayed only after the script generates the final result tree.

Junos OS op scripts, event scripts, and commit scripts prepend "error:" to the displayed message when generating text output. When generating XML output, the scripts place the output inside a **<message>** element, which is enclosed in an **<xmn:error>** element.

If the **message** statement is used in a commit script, the script will generate two errors and terminate the commit process. If the **message** statement is used in an event script, the script writes the message to the output file, if one is configured.

**Attributes** message *expression*—XPath expression or string emitted as output.

**SLAX Example**

```
if (not(valid)) {
  message "The " _ name() _ " node is not valid";
}
```

## mode

**Syntax**

mode *qualified-name*;

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Indicate the mode in which a template needs to be applied for the template to be used. If templates are applied in the specified mode, the **match** statement is used to determine whether the template can be used with the particular node. If more than one template matches a node in the specified mode, the priority statement determines which template is used. The highest priority wins. If no priority is specified explicitly, the priority of a template is determined by the **match** statement.

This statement is comparable to the **mode** attribute of the **<xsl:template>** element. You can include this statement inside a SLAX **match** or **apply-templates** statement.

**SLAX Example**

```
match * {
  mode "one";
  <one> .;
```

```
}

match * {
  mode "two";
  <two> string-length(.);
}

match / {
  apply-templates version {
    mode "one";
  }
  apply-templates version {
    mode "two";
  }
}
```

**XSLT Equivalent**

```
<xsl:template match="*" mode="one">
  <one>
    <xsl:value-of select="."/>
  </one>
</xsl:template>

<xsl:template match="*" mode="two">
  <two>
    <xsl:value-of select="string-length(.)"/>
  </two>
</xsl:template>

<xsl:template match="/">
  <xsl:apply-templates select="version" mode="one"/>
  <xsl:apply-templates select="version" mode="two"/>
</xsl:template>
```

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344.

**Related Documentation**

- apply-templates on page 165
- call on page 169
- match on page 185
- priority on page 196
- template on page 202
- with on page 209
- xsl:template on page 157

## mvar

**Syntax** mvar $*name*[=*initial-value*];

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Declare a mutable variable in a SLAX script. You can initialize a mutable variable when you declare it by following the variable name with an equal sign (=) and a value.

Mutable variables differ from variables declared using the **var** statement in that you can change the value of a mutable variable after it is declared. To initialize or set the value of a mutable variable after you declare it, use the **set** statement. To append a value to the node set contained in a mutable variable, use the **append** statement.

NOTE: **Mutable variables use non-standard SLAX specific extension elements, which can affect the portability of a script.**

**Attributes** *name*—Mutable variable identifier. After declaration, you can reference the variable within expressions by using the identifier prefixed with the dollar sign ($) character.

*initial-value*—Initial value assigned to the mutable variable.

**SLAX Example** The following example creates the mutable variable **block**, and initializes it. The **set** statement assigns a new value to the **block** variable, overwriting the initial value set in the declaration. In the **for** loop, the code iterates over each item in the specified list and appends an **<item>** element with two child elements, **<name>** and **<size>**, to the node set stored in the **block** variable.

```
mvar $block= <block> "start here";
set $block = <block> "item list";

for $item (list) {
  append $block += <item> {
    <name> $item/name;
    <size> $item/size;
  }
}
```

**Related Documentation**

- SLAX Variables Overview on page 53
- append on page 164
- set on page 200
- var on page 207

## number

**Syntax**

```
number expression {
  format numbering-style;
  grouping-separator character;
  grouping-size number;
}

number {
  count nodes;
  format numbering-style;
```

```
  from nodes;
  grouping-separator character;
  grouping-size number;
  level "single" | "multiple" | "any";
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Generate a formatted number string, which is output to the result tree. When used with an argument, the statement formats the number given by that XPath expression. When used without an argument, the statement uses the **count**, **from**, and **level** options to generate the number based on the position of one or more nodes within the current XML document. In both cases, optional statements specify the formatting for that number. If needed, you can also redirect the formatted number string to a variable or output method instead of the result tree.

**Attributes** number *expression*—XPath expression providing the number to format.

count *nodes*—XPath expression specifying which nodes should be counted. If **count** is omitted, it defaults to nodes with the same name as the current node.

format *numbering-style*—A string, variable, or XPath expression that defines the number formatting.

The **format** option can include the following:

- start string–Any non-alphanumeric characters that precede the first number token in the format string. The start string is prepended to the formatted number string.
- number token–One or more number tokens that indicate what numbering format to use for the included numbers. The formatted number string only includes more than one number if the **level** option is set to "multiple". Table 25 on page 189 lists format values and corresponding styles. The default value is "1", which uses a decimal format style. When using decimal format, you can specify the minimum length of the formatted number string by preceding the "1" with one or more zeros.
- token separator–Non-alphanumeric characters that separate number tokens in the format string. These characters are included in the formatted number string between the computed numbers.
- end string–Any non-alphanumeric characters that follow the last number token in the format string. The end string is appended to the formatted number string.

### Table 25: Numbering Styles for SLAX Statement number, format Option

| Format Value | Style | Example |
|---|---|---|
| 1 | Decimal format | 1 2 3 ...10 11 ... |
| 01 | Decimal format with a minimum output string length of 2 | 01 02 03 ... 10 11 ... |

Table 25: Numbering Styles for SLAX Statement number, format Option *(continued)*

| Format Value | Style | Example |
|---|---|---|
| 001 | Decimal format with a minimum output string length of 3 | 001 002 003 ... 010 011 012 ... 100, 101 |
| a | Lowercase alphabetic numbering | a b c ... z ... aa ab ... az ... ba bb ... |
| A | Uppercase alphabetic numbering | A B C ... Z ... AA AB ... BA BB ... |
| i | Lowercase Roman numbering | i ii iii iv v ... |
| I | Uppercase Roman numbering | I II III IV V ... |

from *nodes*—XPath expression specifying from which element to start the count. When **level** is set to **single** or **multiple**, this option constrains the counting to only node descendants of the nearest ancestor that matches the expression. When **level** is set to **any**, this option constrains the counting to only nodes that follow the nearest ancestor or preceding node of the current node that matches the expression.

grouping-separator *character*—Character used to delimit groups of digits for numbers expressed in decimal format. For example, decimal notation uses a comma as the delimiter between digit groupings.

grouping-size *number*—Defines the number of digits in a group for numbers expressed in decimal format. Setting this option causes the formatted number to be split into multiple groups according to the grouping size, with the grouping separator delimiting the groups. For example, decimal notation often uses a grouping size of 3.

level—Specifies what type of counting to perform. Accepted values are **single**, **multiple**, and **any**. The default is **single**. Specifying **single** starts the counting from the first ancestor node, specifying **multiple** starts the counting from any ancestor node, and specifying **any** starts the counting from any node.

- **single**–Perform only one count. The current node, if it matches the count expression, or the nearest ancestor that matches the count expression, is counted. The position of the node in document order, relative to its siblings that also match the count parameter, is used as the number to be formatted.
- **multiple**–Separately count all nodes that match the count expression and are either the current node or an ancestor of the current node. The position of each node in document order, relative to its siblings that also match the count parameter, is used as one of the numbers to be formatted.
- **any**–Perform only one count. The current node, if it matches the count pattern, or its nearest ancestor or preceding node that matches the count pattern, is counted. The position of the node in document order, relative to all other matching nodes that are ancestors or precede the node, is used as the number to be formatted.

NOTE: Currently libxslt (1.1.26) does not support the “language” and “letter-value” options for the <xsl:number> element. While SLAX provides a means of encoding these XSLT constructs, they are not usable under Junos OS.

**SLAX Example**

The following sample code iterates from 1 through 5. For each integer, the **number** statement outputs the equivalent uppercase Roman numeral value.

```
for $i (1 ... 5) {
 number $i {
  format "I ";
 }
}
```

```
I II III IV V
```

The following sample code provides the string “1234567890” to the **number** statement, which formats the output in decimal format with a group size of 3 and a comma as a group delimiter.

```
number "1234567890" {
  grouping-size 3;
  grouping-separator ",";
  format "1";
}
```

```
1,234,567,890
```

The following sample code counts all the **name** elements in the configuration hierarchy stored in the variable **$data**. The **count** option combined with the **level "multiple"** option tracks the count for any **name** elements under the **interface**, **unit**, and **address** elements.

The **format** option (1.A.a) includes a start string, which is an open parenthesis, and an end string, which is a close parenthesis and a space character. The number tokens are “1”, “A”, and “a”, which define the formatting of the numbers as decimal format, uppercase alphabetic numbering, and lowercase alphabetic numbering, respectively. The token separator is a period, which is also included in the output.

```
var $data := {
 <interfaces> {
   <interface> {
     <name> "ge-0/0/0";
     <unit> {
       <name> "0";
     }
     <unit> {
       <name> "1";
     }
   }
    <interface> {
     <name> "ge-0/1/0";
     <unit> {
```

```
        <name> "10";
        <family> {
          <inet>;
        }
      }
    }
    <interface> {
      <name> "ge-2/0/2";
      <unit> {
        <name> "0";
        <family> {
          <inet> {
            <address> {
              <name> "10.1.1.1/24";
            }
          }
        }
      }
    }
  }
}

for-each ($data//name) {
  number {
   level "multiple";
   count interface|unit|address;
   format "(1.A.a) ";
  }
  expr . _ "\n";
}
```

For the generated numbers displayed in the result tree, the decimal number in parentheses is associated with a particular interface. For each interface, the uppercase letter is associated with each logical unit name, and any lowercase letter is associated with the address **name** element for that logical unit, which is the IP address.

```
(1) ge-0/0/0
(1.A) 0
(1.B) 1
(2) ge-0/1/0
(2.A) 10
(3) ge-2/0/2
(3.A) 0
(3.A.a) 10.1.1.1/24
```

**Related Documentation**

- decimal-format on page 171
- output-method on page 192

## output-method

**Syntax**

```
output-method output-format {
  cdata-section-elements name-list;
  doctype-public string;
  doctype-system string;
  encoding string;
```

```
    indent "yes" | "no";
    media-type string;
    omit-xml-declaration "yes" | "no";
    standalone "yes" | "no";
    version string;
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Define the style used for result tree output. The **output-method** statement must be defined as a top-level statement in the script. Output formats include HTML, text, or XML. The default is XML, unless the first child element of the root node is **<html>** and there are no preceding text nodes, in which case the default output format is HTML.

**Attributes** *output-format*—Specify the format of the output. Acceptable values are “html”, “text”, “xml”, or a qualified name. The default is XML, unless the first child element of the root node is **<html>** and there are no preceding text nodes, in which case the default output format is HTML. Specifying a format of XML adds the XML declaration (**<?xml ?>**) to the result tree file.

cdata-section-elements *name-list*—Specify a space-delimited list of the names of output elements whose text contents should be output to the result tree using CDATA sections. A CDATA section starts with "<![CDATA[" and ends with "]]>", and the contents of the section are interpreted by an XML parser as character data only, rather than markup.

doctype-public *string*—Add the DOCTYPE declaration to the result tree, and specify the value of the **PUBLIC** attribute, which tells the parser where to locate the Document Type Definition (DTD) file.

doctype-system *string*—Add the DOCTYPE declaration to the result tree, and specify the value of the **SYSTEM** attribute, which tells the parser where to locate the DTD file on the system.

encoding *string*—Explicitly add the pseudo-attribute **encoding** to the XML declaration in the output, and specify the character encoding used to encode the document, for example UTF-8, UTF-16, or ISO-8859-1.

indent "yes" | "no"—Specify whether to indent the result tree output according to the hierarchical structure. Acceptable values are “yes” and “no”. The default is no indentation.

media-type *string*—Define the MIME content type of the output. The default is "text/xml".

omit-xml-declaration "yes" | "no"—Specify whether to include or omit the XML declaration (**<?xml ?>** ) in the output. The default is "no".

standalone "yes" | "no"—Explicitly add the pseudo-attribute **standalone** with the given string value to the XML declaration (**<?xml ?>** ) in the output . Acceptable values are “yes” and “no”. The **standalone** attribute is only relevant if the document uses a

DTD. If the **standalone** option is not included in the **output-method** statement, there is no explicit declaration in the result tree, which is identical to **standalone="no"**.

version *string*—For HTML and XML formats, set the W3C version for the output format. The pseudo-attribute **version** is included in the XML declaration ( **<?xml ?>** ) with the given version number.

**SLAX Example** The following example uses the output method XML, which creates an XML declaration in the result tree output and adds the pseudo-attributes **version**, **encoding**, and **standalone** to the declaration. The DOCTYPE declaration has the root element **<html>** and provides values for both the **PUBLIC** and the **SYSTEM** attributes.

```
version 1.1;

output-method xml {
  doctype-public "-//W3C//DTD XHTML 1.0 Transitional//EN";
  doctype-system "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd";
  encoding "utf-8";
  indent "yes";
  omit-xml-declaration "no";
  standalone "no";
  version "1.0";
}

match / {
  <html> {
    <script type="text/javascript" src="/assets/js/api.js">;
   /* ... */
  }
}
```

The script produces the following output:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <script type="text/javascript" src="/assets/js/api.js"></script>
...
</html>
```

The following example is similar to the previous example except that the script does not specify an output format. Since the first child element of the root node is **<html>**, the output format defaults to HTML.

```
version 1.1;

output-method {
  doctype-public "-//W3C//DTD XHTML 1.0 Transitional//EN";
  doctype-system "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd";
  encoding "utf-8";
  indent "yes";
  omit-xml-declaration "no";
  standalone "no";
  version "1.0";
}
```

```
match / {
  <html> {
    <script type="text/javascript" src="/assets/js/api.js">;
  /* ... */
  }
}
```

The default output format is HTML. The XML declaration is omitted from the output.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><script type="text/javascript" src="/assets/js/api.js"></script></html>
```

## param

**Syntax** param $*name*=*value*;

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Declare a parameter for a template or for the style sheet as a whole. Template parameters declared with the **param** statement must be placed inside the template code block. A global parameter, the scope of which is the entire style sheet, must be declared at the top level of the style sheet. You can include an initial value by following the parameter name with an equal sign (**=**) and a value expression. A parameter whose value is set by Junos OS at script initialization must be defined as a global parameter.

In SLAX, parameter and variable names are declared and accessed using the dollar sign ($). This is unlike the **name** attribute of **<xsl:variable>** and **<xsl:parameter>** elements, which do not include the dollar sign in the declaration.

**Attributes** *name*—Defines the name of the parameter.

*value*—Defines the default value for the parameter, which is used if the person or client application that executes the script does not explicitly provide a value.

**SLAX Example**

```
param $vrf;
param $dot = .;
```

**XSLT Equivalent**

```
<xsl:param name="vrf"/>
<xsl:param name="dot" select="."/>
```

**Usage Examples**

- Example: Imposing a Minimum MTU Setting on page 395
- Example: Limiting the Number of ATM Virtual Circuits on page 397
- Example: Limiting the Number of E1 Interfaces on page 401
- Example: Preventing Import of the Full Routing Table on page 429
- Example: Requiring and Restricting Configuration Statements on page 435

**Related Documentation**

- SLAX Parameters Overview on page 49

- SLAX Templates Overview on page 44
- template on page 202
- var on page 207
- with on page 209

## preserve-space

**Syntax** preserve-space *element-list*;

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Preserve whitespace-only child text nodes for the source tree element nodes listed, but not for the child text nodes of the element node children. To preserve whitespace-only child text nodes of the element node children, specify the child nodes as separate entries in the preserve-space element list. Specifying an asterisk preserves whitespace-only child elements for all elements, which is the default behavior. A text node is considered whitespace-only if it includes only spaces, tabs, newlines, and carriage returns.

The **preserve-space** statement is only needed if the **strip-space** statement has been used with an asterisk, indicating that whitespace-only child text nodes should be removed from all element nodes. In this case, use the **preserve-space** statement to indicate specific element nodes that should not have their whitespace-only child text nodes stripped.

This statement must be defined as a top-level statement in the script.

**Attributes** *element-list*—Space-separated list of element names for which to preserve whitespace-only child text nodes.

**SLAX Example** The following example removes all whitespace-only text nodes from the source tree except for child elements of **<user-context>**:

```
version 1.1;

preserve-space user-context;
strip-space *;

match / {
...
}
```

**Related Documentation**
- strip-space on page 202

## priority

**Syntax** priority *number*;

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** If more than one template matches a node in the specified mode, this statement determines which template is used. The highest priority wins. If no priority is specified explicitly, the priority of a template is determined by the **match** statement.

This statement is comparable to the **priority** attribute of the **<xsl:template>** element. You can include this statement inside a SLAX **match** statement.

**SLAX Example**

```
match * {
  priority 10;
  <output> .;
}
```

**XSLT Equivalent**

```
<xsl:template match="*" priority="10">
  <output>
    <xsl:value-of select="."/>
  </output>
</xsl:template>
```

**Usage Examples** None of the examples in this manual use this statement.

**Related Documentation**

- apply-templates on page 165
- call on page 169
- match on page 185
- mode on page 186
- template on page 202
- with on page 209
- xsl:template on page 157

## processing-instruction

**Syntax**

processing-instruction *instruction-name*;

processing-instruction *instruction-name* {
  *instruction-value*;
}

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Add an XML processing instruction to the result tree. A processing instruction is a mechanism to convey application-specific information inside an XML document. The application can detect processing instructions and change their behavior accordingly. The instruction name is mandatory and becomes the target of the processing instruction. It can be a hard-coded string, a variable, or an XPath expression. The optional body generates the processing instruction's content, which consists of one or more name-value pairs. The generated instruction is enclosed within the tags **<?** and **?>**.

Junos OS SLAX scripts generally do not require the **processing-instruction** statement, because the result tree is processed directly by Junos OS. However, you might add a processing instruction to an XML document that is written to disk through the **<xsl:document>** instruction element or one of its related extension elements.

**Attributes** *instruction-name*—Identifier for the processing instruction, which can be a string, a variable, or an XPath expression.

*instruction-value*—Instruction content, which consists of name-value pairs.

**SLAX Example** The following code creates the processing instruction **xml-stylesheet**. The instruction content contains two name-value pairs: **type** and **href**.

```
processing-instruction "xml-stylesheet" {
  expr 'type="text/css" ';
  expr 'href="style.css"';
}
```

The corresponding output in the result tree is:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

The following example writes an XML document to the file **/var/tmp/output.xml** using the **<xsl:document>** instruction element. The script adds a processing instruction named **instruction** to the document.

```
version 1.1;

match / {
  <op-script-results> {
    <xsl:document href="/var/tmp/output.xml" indent="yes" method="xml"> {
      <document-element> {
        <element>;
        processing-instruction "instruction" {
          expr 'name="testing"';
        }
        <element>;
      }
    }
  }
}
```

The script generates the file **/var/tmp/output.xml**, which contains the processing instruction enclosed within **<?** and **?>** tags.

```
<?xml version="1.0"?>
<document-element>
    <element/>
    <?instruction name="testing"?>
    <element/>
</document-element>
```

## result

**Syntax**

```
result expression;

result {
 /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Define the return value for a function. The value can be a simple scalar value, an XML element or XPath expression, or a set of instructions that emit the value to be returned.

**Attributes** result *expression*—XPath expression defining the return value of the function.

**SLAX Example** The following example defines three extension functions, **my:size()**, **my:box-parts()**, and **my:ark()**. The **my:ark()** function returns a node set containing an **<ark>** element that encloses the node set returned by the **my:box-parts()** function. The **my:box-parts()** function returns a node set containing a **<box>** element enclosing three **<part>** child elements. The content of each **<part>** element is the value returned by the **my:size()** function. The return value of the **my:size()** function is the product of the three parameters **width**, **height**, and **scale**.

```
version 1.1;
ns my  exclude = "http://www.example.com/myfunctions";

function my:size ($x, $y, $scale = 1) {
  result $x * $y * $scale;
}
function my:box-parts ($width, $height, $depth, $scale = 1) {
  result <box> {
    <part count=2> my:size($width, $depth);
    <part count=2> my:size($width, $height);
    <part count=2> my:size($depth, $height);
  }
}
function my:ark () {
  result {
    <ark> {
      copy-of my:box-parts(2.5, 1.5, 1.5);
    }
  }
}

match / {
  var $res = my:ark();
  copy-of $res;
}
```

**Related Documentation**

- SLAX Functions Overview on page 47
- copy-of on page 171

- function on page 180

## set

**Syntax**

```
set $name = value;
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Assign a value to a mutable variable. The variable must be defined using the **mvar** statement.

**Attributes** *name*—Name of the mutable variable.

*value*—Value to assign to the mutable variable.

**SLAX Example** The following example creates the mutable variable, **block**. The **set** statement assigns an initial value to the **block** variable. In the **for** loop, the code iterates over each item in the specified list and appends an **<item>** element with two child elements, **<name>** and **<size>**, to the node set stored in the **block** variable.

```
mvar $block;
set $block = <block> "item list";

for $item (list) {
  append $block += <item> {
    <name> $item/name;
    <size> $item/size;
  }
}
```

**Related Documentation**

- SLAX Variables Overview on page 53
- append on page 164
- mvar on page 187
- var on page 207

## sort

**Syntax**

```
sort expression {;
  case-order "upper-first" | "lower-first";
  data-type "text" | "number" | type-name;
  order "ascending" | "descending";
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Control the order in which the **for-each** and **apply-templates** statements iterate through the current node list. By default, the **for-each** and **apply-templates** statements consider nodes in document order, but the **sort** statement defines the order prior to iterating

through the node list. Insert the **sort** statement immediately after the **for-each** or **apply-templates** statement. The **sort** statement is only processed when the loop is first initiated.

The **sort** statement has an optional XPath expression and three optional parameters: **case-order**, **data-type**, and **order**. The XPath expression determines each node's comparison string used for sorting. The script evaluates the expression with the node as its context, and then translates the result into the comparison string for that node. If you do not specify an XPath expression, the default value is ".", which causes the string content of each node in the list to be compared. SLAX-specific operators such as == and _ cannot be used within the expression string. If the **sort** statement does not include any optional parameters, the list is sorted based on the string value of each node.

The **sort** statement does not permanently sort the underlying XML data structure, only the order of the current node list being used by the **for-each** or **apply-templates** statement. Multiple **sort** statements can be assigned to a single **for-each** or **apply-templates** statement. They are applied, in order, until a difference is found.

**Attributes**

*expression*—XPath expression that determines each node's comparison string used for sorting. The default value is ".".

case-order—Specify whether to sort lowercase first or uppercase first. Acceptable values are "lower-first" or "upper-first". The default is "upper-first".

data-type—Specify the element type, which determines whether a numerical, lexical, or other sort is performed. Acceptable values are "number" and "text". The default is "text".

Setting **data-type** to "text" compares the strings based on their character values (that is ASCII code), so "0" is less than "9", which is less than "A", which is less than "Z", which is less than "a", which is less than "z". Setting **data-type** to "number" converts the strings to numbers and compares them numerically. With ascending text sorting, "100" would come before "11" because "0" has a lower ASCII code than "1", but with ascending number sorting, 11 would come before 100 because 11 is a smaller number than 100.

order—Specify whether to sort in ascending or descending order. Acceptable values are "descending" or "ascending". The default is "ascending".

**SLAX Example**

The following example SLAX script executes the Junos XML API **get-interface-information** command and parses the resulting output. The **for-each** loop prints the name of each physical interface on the device sorted in ascending order.

```
version 1.1;
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";

match / {
  <op-script-results> {

    var $results = jcs:invoke("get-interface-information");
```

```
    for-each ($results/physical-interface/name) {
      sort . {
        data-type "text";
        order "ascending";
      }
      <interface-name> .;
    }
  }
}
```

**Related Documentation**

- apply-templates on page 165
- for-each on page 178

## strip-space

**Syntax**

```
strip-space element-list;
```

**Release Information**

Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description**

Remove whitespace-only child text nodes from the source tree element nodes listed, but not from the child text nodes of the element node children. To perform whitespace stripping on the child text nodes of the element node children, specify the child nodes as separate entries in the strip-space element list. Specifying an asterisk removes whitespace-only child elements from all elements. A text node is considered whitespace-only if it includes only spaces, tabs, newlines, and carriage returns.

This statement must be defined as a top-level statement in the script. The default is to preserve all whitespace-only elements.

**Attributes**

*element-list*—List of element names separated by spaces.

**SLAX Example**

The following example removes all whitespace-only text nodes from the source tree except for child elements of **<user-context>**:

```
version 1.1;

preserve-space user-context;
strip-space *;

match / {
...
}
```

**Related Documentation**

- preserve-space on page 196

## template

**Syntax**

```
template qualified-name (parameter-name = value) {
  /* code */
}
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Declare a named template. You can include a comma-separated list of parameter declarations, with the parameter name and an optional equal sign (=) and value expression. You can declare additional parameters inside the code block using the **param** statement. You can invoke the template using the **call** statement.

**SLAX Example**

```
match configuration {
  var $name-servers = name-servers/name;
  call temp();
  call temp($name-servers, $size = count($name-servers));
  call temp() {
    with $name-servers;
    with $size = count($name-servers);
  }

  template temp($name-servers, $size = 0) {
    <output> "template called with size " _ $size;
  }
}
```

**XSLT Equivalent**

```
<xsl:template match="configuration">
  <xsl:variable name="name-servers" select="name-servers/name"/>
  <xsl:call-template name="temp"/>
  <xsl:call-template name="temp">
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
  <xsl:call-template name="temp">
    <xsl:with-param name="name-servers" select="$name-servers"/>
    <xsl:with-param name="size" select="count($name-servers)"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="temp">
  <xsl:param name="name-servers"/>
  <xsl:param name="size" select="0"/>
  <output>
    <xsl:value-of select="concat('template called with size ', $size)"/>
  </output>
</xsl:template>
```

**Usage Examples** See "Example: Adding a Final then accept Term to a Firewall" on page 344 and "Example: Adding T1 Interfaces to a RIP Group" on page 348.

**Related Documentation**

- SLAX Parameters Overview on page 49
- SLAX Templates Overview on page 44
- apply-templates on page 165
- call on page 169
- match on page 185

- mode on page 186
- priority on page 196
- with on page 209

## terminate

**Syntax**

terminate *expression*;

```
terminate {
  /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Generate an error message that is immediately displayed to the user, and exit the script.

Junos OS op scripts, event scripts, and commit scripts prepend "error:" to the displayed message when generating text output. When generating XML output, the scripts place the output inside a **<message>** element, which is enclosed in an **<xmn:error>** element.

If the **terminate** statement is used in a commit script, the script will generate two errors and terminate the script and the commit process. If the **terminate** statement is used in an event script, the script writes the message to the output file, if one is configured, and terminates the script.

**Attributes** *expression*—XPath expression or string emitted as output.

**SLAX Example**

```
if (not(valid)) {
  terminate "The " _ name() _ " node is not valid. Exiting script.";
}
```

**Related Documentation**

- message on page 186

## trace

**Syntax**

trace *expression*;

```
trace {
  /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Write a message to the trace file, if tracing is enabled. If tracing is not enabled, trace output is not generated. The **trace** message can be an XPath expression or string, or it can be generated by the contents of a **trace** statement block.

Enabling of tracing is typically a feature of the environment in which a SLAX script is called. When executing a script using the **slaxproc** command, include the **--trace** or **-t** option to enable tracing. For more information about slaxproc, see "Understanding the SLAX Processor (slaxproc)" on page 80.

**Attributes** trace *expression*—XPath expression or string written to the trace file.

**SLAX Example** The following examples demonstrate the **trace** statement syntax. The first example writes a concatenated string to the trace file. The second example uses a code block to output a **<max>** element and a **<min>** element and the values of the **max** and **min** variables. The third example uses a conditional statement to specify when to output trace data. If the expression evaluates to **true**, the code block writes the string and the **<options>** element hierarchy to the trace file.

```
trace "max " _ $max _ "; min " _ $min;

trace {
  <max> $max;
  <min> $min;
}

trace {
  if ($my-trace-flag) {
    expr "max " _ $max _ "; min " _ $min;
    copy-of options;
  }
}
```

**Related Documentation**

- message on page 186
- terminate on page 204
- Understanding the SLAX Processor (slaxproc) on page 80

## uexpr

**Syntax** uexpr *expression*;

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Generate the string value of an XPath expression and add it to the result tree, but do not escape special characters. The **uexpr** statement behaves identically to the **expr** statement, except that the contents are not escaped. By default, characters such as "<", ">", and "&" are escaped into proper XML as "<", ">", and "&", respectively, but **uexpr** does not execute this escaping mechanism.

**Attributes** *expression*—XPath expression to add to the result tree.

**SLAX Example** The following statement outputs the string to the result tree exactly as it appears in the statement. If **expr** is used in place of **uexpr**, the script would output the string "<:&>".

```
uexpr "<:-&>";
```

**Related Documentation**

- expr on page 175
- message on page 186
- terminate on page 204
- trace on page 204

## use-attribute-sets

**Syntax** use-attribute-sets *attribute-set-name*;

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Add the attributes in the attribute set to the current element. The **use-attribute-sets** statement can be used under the **attribute-set**, **copy-node**, and **element** statements, as well as under a normal element.

**Attributes** *attribute-set-name*—Name of the attribute set, which is defined using an **attribute-set** statement.

**SLAX Example** The following example creates two attribute sets: **table-attributes** and **table-attributes-ext**. The **table-attributes-ext** set includes all of the attributes that are already defined in the **table-attributes** set through use of the **use-attributes-sets** statement. In the main script body, the **table-attributes-ext** attribute set is applied to the **<table>** element. The **<table>** element includes the four attributes: **order**, **cellpadding**, **cellspacing**, and **border**.

```
version 1.1;

var $cellpadding = "0";
var $cellspacing = "10";

attribute-set table-attributes {
  attribute "order" { expr "0"; }
  attribute "cellpadding" { expr $cellpadding; }
  attribute "cellspacing" { expr $cellspacing; }
}
attribute-set table-attributes-ext {
  use-attribute-sets table-attributes;
  attribute "border" { expr "0"; }
}

match / {
  ...
  <table> {
    use-attribute-sets table-attributes-ext;
```

```
  }
}
```

**Related Documentation**

- SLAX Elements and Element Attributes Overview on page 42
- attribute on page 166
- attribute-set on page 167
- element on page 173

## var

**Syntax**

```
var $name=value;
```

**Release Information**

Statement introduced in version 1.0 of the SLAX language.

**Description**

Declare a local or global variable. A variable is global if it is defined outside of any template. Otherwise, it is local. The value of a global variable is accessible anywhere in the style sheet. The scope of a local variable is limited to the template or code block in which it is defined. Variables declared in this manner are immutable. You initialize a variable by following the variable name with an equal sign (=) and an expression.

**Attributes**

*name*—Specifies the name of the variable. After declaration, the variable can be referred to within expressions using this name, including the $ character.

*value*—Defines the default value for the variable, which is used if the person or client application that executes the script does not explicitly provide a value.

**SLAX Example**

```
var $vrf;
var $location = $dot/@location;
var $message = "We are in "_ $location _" now.";
```

**XSLT Equivalent**

```
<xsl:variable name="vrf"/>
<xsl:variable name="location" select="$dot/location"/>
<xsl:variable name="message" select="concat('We are in ', $location, now.')"/>
```

**Usage Examples**

See "Example: Limiting the Number of E1 Interfaces" on page 401, "Example: Limiting the Number of ATM Virtual Circuits" on page 397, "Example: Configuring Administrative Groups for LSPs" on page 362, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Variables Overview on page 53
- mvar on page 187
- param on page 195

## version

**Syntax**

```
version 1.0;
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Specify the version of SLAX that is being used. All SLAX style sheets must begin with a **version** statement.

Version 1.0 uses XML version 1.0 and XSLT version 1.1.

In addition, the **xsl** namespace is implicitly defined as follows:

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

**Attributes** *version-number*—Specifies the version of SLAX. Junos OS supports SLAX version 1.0.

**SLAX Example**

```
version 1.0;
```

**XSLT Equivalent**

```
<xsl:stylesheet version="1.0">
```

**Usage Examples**

- Example: Adding a Final then accept Term to a Firewall on page 344
- Example: Changing the Configuration Using an Op Script on page 491
- Example: Assigning a Classifier on page 351
- Example: Imposing a Minimum MTU Setting on page 395
- Example: Restarting an FPC Using an Op Script on page 528

**Related Documentation**

- Required Boilerplate for Commit Scripts on page 270
- Required Boilerplate for Event Scripts on page 665
- Required Boilerplate for Op Scripts on page 463
- SLAX Syntax Rules Overview on page 39

## while

**Syntax**

```
while (expression) {
  /* body */
}
```

**Release Information** Statement introduced in version 1.1 of the SLAX language, which is supported in Junos OS Release 12.2 and later releases.

**Description** Repeatedly execute a block of statements until the specified condition evaluates to **false**. The condition is an XPath expression that is converted to a boolean type. If the expression evaluates to **true**, the contents of the while loop are executed. The loop continues to execute until the expression evaluates to **false**. During execution, the context is not changed. In the expression, you should use a mutable variable, which is declared using the **mvar** statement, to avoid creating an infinite loop.

**Attributes** *expression*—XPath expression, which is cast to boolean type and used as the condition for the while loop. The code block contents are executed as long as the condition evaluates to **true**.

**SLAX Example** In the example, the while loop parses through the item list until the desired value is found. When that value is detected, **$seen** is set to true, and the while loop exits.

```
mvar $seen = false();
mvar $count = 1;

while (not($seen)) {
  if (item[$count]/value) {
    set $seen = true();
  }
  set $count = $count + 1;
}
```

**Related Documentation**

- SLAX Variables Overview on page 53
- XPath Overview on page 22
- mvar on page 187

## with

**Syntax**

```
with $name = value;
```

**Release Information** Statement introduced in version 1.0 of the SLAX language.

**Description** Specify a parameter to pass into a template. You can use this statement when you apply templates with the **apply-templates** statement or invoke templates with the **call** statement.

Optionally, you can specify a value for the parameter by including an equal sign (**=**) and a value expression. If no value is specified, the current value of the parameter is passed to the template.

**Attributes** *name*—Name of the variable or parameter for which the value is being passed.

*value*—Value of the parameter being passed to the template.

**SLAX Example**

```
match configuration {
  var $domain = domain-name;
  apply-templates system/host-name {
    with $message = "Invalid host-name";
    with $domain;
  }
}

match host-name {
  param $message = "Error";
  param $domain;
  <hello> $message _ ":: " _ . _ " (" _ $domain _ ")";
}
```

**XSLT Equivalent**

```
<xsl:template match="configuration">
  <xsl:apply-templates select="system/host-name">
    <xsl:with-param name="message" select="'Invalid host-name'"/>
```

```
    <xsl:with-param name="domain" select="$domain"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="host-name">
  <xsl:param name="message" select="'Error'"/>
  <xsl:param name="domain"/>
  <hello>
    <xsl:value-of select="concat($message, ':: ', ., ' (', $domain, ')')"/>
  </hello>
</xsl:template>
```

**Usage Examples**

See "Example: Configuring Dual Routing Engines" on page 370, "Example: Preventing Import of the Full Routing Table" on page 429, and "Example: Automatically Configuring Logical Interfaces and IP Addresses" on page 355.

**Related Documentation**

- SLAX Parameters Overview on page 49
- SLAX Templates Overview on page 44
- apply-templates on page 165
- call on page 169
- match on page 185
- mode on page 186
- priority on page 196
- template on page 202

PART 2

# Junos Script Automation

This section contains information common to all script types.

CHAPTER 10

# Storing, Enabling, and Updating Scripts



## Storing and Enabling Scripts

To use a script on a switch, router, or security device, you must copy the script to the device and enable it in the configuration.

1. Create the script.
2. Copy the script to the appropriate directory on the device for that script type. Only users who belong to the Junos OS **super-user** login class can access and edit files in the script directories on a device running Junos OS.

   By default, scripts are stored in and executed from the **/var/db/scripts** directory on the device's hard drive under the subdirectory appropriate to the script type. You can also store scripts on the flash drive in the **/config/scripts** directory under the subdirectory appropriate to the script type.

   **commit script**—Copy the script to the **/var/db/scripts/commit** directory on the hard drive or the **/config/scripts/commit** directory on the flash drive.

   **op script**—Copy the script to the **/var/db/scripts/op** directory on the hard drive or the **/config/scripts/op** directory on the flash drive.

   **event script**—Copy the script to the **/var/db/scripts/event** directory on the hard drive or the **/config/scripts/event** directory on the flash drive.

NOTE: If the device has dual Routing Engines and you want to enable the script to execute on both Routing Engines, you must copy it to the appropriate directory on both Routing Engines. The **commit synchronize** command does not automatically copy scripts between Routing Engines.

3. Enable the script by including the **file *filename*** statement at the appropriate hierarchy level for that script type.

   **commit script**—Include the **file *filename*** statement at the **[edit system scripts commit]** hierarchy level. For instructions, see "Controlling Execution of Commit Scripts During Commit Operations" on page 279.

   **op script**—Include the **file *filename*** statement at the **[edit system scripts op]** hierarchy level. For instructions, see "Enabling an Op Script and Defining a Script Alias" on page 474.

   **event script**—Include the **file *filename*** statement at the **[edit event-options event-script]** hierarchy level. For instructions, see "Enabling an Event Script" on page 676.

4. If you store scripts in and load them from flash memory, include the **load-scripts-from-flash** statement at the **[edit system scripts]** hierarchy level. For detailed information about storing scripts in flash memory, see "Storing Scripts in Flash Memory" on page 214.

   ```
   [edit]
   user@host# set system scripts load-scripts-from-flash
   ```

5. Issue the **commit** command.

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

Newly enabled commit scripts do not execute during the commit operation but execute automatically during each subsequent commit operation. After the commit operation completes, enabled event scripts are loaded into memory and are ready for automatic execution in response to system log events. For more information, see "Executing Event Scripts in an Event Policy" on page 578. After the commit operation completes, op scripts can be executed on the device. For more information, see "Executing an Op Script" on page 476.

**Related Documentation**

- Storing Scripts in Flash Memory on page 214
- Storing and Using Imported Scripts and Script Functionality on page 215
- Controlling Execution of Commit Scripts During Commit Operations on page 279

## Storing Scripts in Flash Memory

By default, scripts are stored in and executed from the **/var/db/scripts/** directory on the device's hard drive under the subdirectory appropriate to the script type. To store scripts in and load them from flash memory instead, include the **load-scripts-from-flash** statement at the **[edit system scripts]** hierarchy level:

```
[edit]
user@host# set system scripts load-scripts-from-flash
```

When you add the **load-scripts-from-flash** statement in the configuration, all commit, event, operation, and script library scripts are loaded from the **/config/scripts/** directory on the flash drive under the subdirectory appropriate to the script type. You must manually

move scripts from the hard drive to the flash drive. They are not moved automatically. Similarly, if you delete the **load-scripts-from-flash** statement from the configuration, you must manually copy the scripts from the flash drive to the hard drive to ensure that the current versions of the scripts are executed. Changing the scripts' physical location has no effect on their operation.

The **/var/run/scripts/** directory always links to the directory from which the scripts are loaded. If you do not set the **load-scripts-from-flash** statement in the configuration, **/var/run/scripts/** points to the **/var/db/scripts/** directory on the device's hard drive. If you set the **load-scripts-from-flash** statement in the configuration, **/var/run/scripts/** points to the **/config/scripts/** directory in flash memory.

To view the scripts currently on the device, list the contents of **/var/run/scripts/*type*/**, where *type* is the subdirectory appropriate to the script type. In the following example, the **load-scripts-from-flash** statement is not configured. In this case, **/var/run/scripts/commit/** points to the**/var/db/scripts/commit/** directory on the hard drive. Listing the files for **/var/run/scripts/commit/** is identical to listing the files in the **/var/db/scripts/commit/** directory.

```
user@host>file list /var/run/scripts/commit

/var/run/scripts/commit:
commit-changes-load-replace.slax
commit-protect.slax

user@host> file list /var/db/scripts/commit

/var/db/scripts/commit:
commit-changes-load-replace.slax
commit-protect.slax

user@host> file list /config/scripts/commit

/config/scripts/commit:
```

**Related Documentation**

- Storing and Enabling Scripts on page 213
- Storing and Using Imported Scripts and Script Functionality on page 215

## Storing and Using Imported Scripts and Script Functionality

Starting with Junos OS Release 11.1, Junos OS provides a dedicated directory for script libraries, where users can store scripts and script functionality that then can be imported into any commit, event, or op script. Upon installation, Junos OS creates the **/var/db/scripts/lib/** directory. Junos OS will not overwrite or erase any files in an existing **lib/** directory upon installation or upgrade.

If you store scripts in and run them from flash memory, both the executed scripts and the imported scripts must be present on the flash drive. When you configure the **load-scripts-from-flash** statement at the **[edit system scripts]** hierarchy level, Junos OS creates the **/config/scripts/lib/** directory. When you add or remove the **load-scripts-from-flash** statement in the configuration, you must manually move scripts

and script libraries from the hard drive to the flash drive, or vice versa, as appropriate. They are not moved automatically.

Imported scripts must be stored in the **/var/db/scripts/lib/** directory on the hard drive, or if the **load-scripts-from-flash** statement is configured, in the **/config/db/scripts/lib/** directory on the flash drive. To import a script from the script library, include the **<xsl:import>** tag in the style sheet declaration of an XSLT script or the **import** statement in a SLAX script and specify the file location. The following sample code imports the **/var/db/scripts/lib/test.xsl** file:

**XSLT Syntax**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0">
  <xsl:import href="../lib/test.xsl"/>
  ...
</xsl: stylesheet>
```

**SLAX Syntax**

```
version 1.0;
import "../lib/test.xsl";
```

**Related Documentation**

- Storing and Enabling Scripts on page 213
- Storing Scripts in Flash Memory on page 214

## Updating Scripts from a Remote Source

- Overview of Updating Scripts from a Remote Source on page 216
- Using a Master Source Location for a Script on page 218
- Using an Alternate Source Location for a Script on page 222

### Overview of Updating Scripts from a Remote Source

You can update the scripts on a device running Junos OS by retrieving a copy from a remote machine (which can be another device running Junos OS or a regular networked computer). This eases file management, because you can make changes to the master script in a single location and then update the copy on each device where the script is currently enabled. Each device continues to use its locally stored scripts, only updating a script when you issue the appropriate operational or configuration mode command.

For each script, you can configure the **source** statement and a URL at the hierarchy level where you configured the script to define the remote location that houses the master copy of that script. When you then issue the **set refresh** configuration mode command for a script, the device running Junos OS updates its local copy by retrieving the remote master copy from that URL.

You can also store a copy of a particular script at a remote location other than the master source. This is convenient when, for example, the master source cannot be accessed due to network issues or other problems. To refresh a single script or multiple scripts from the remote location, you issue the **set refresh-from** configuration mode command at the appropriate hierarchy level and specify the URL. You can also refresh a single script from

a remote location using the **request system scripts refresh-from** operational mode command.

You can use the **set refresh** and **set refresh-from** commands to update either an individual script or multiple scripts of a given type on the device. When you issue the **set refresh** or **set refresh-from** command, the switch, router, or security device immediately attempts to connect to the appropriate remote source for each script. If successful, the device updates the local script with the remote source. If a problem occurs, a set of error messages is returned.

Issuing the **set refresh** or **set refresh-from** command does not add the **refresh** and **refresh-from** statements to the configuration. Thus, these commands behave like operational mode commands by executing an operation, instead of adding a statement to the configuration. The **refresh** and **refresh-from** statements are mutually exclusive.

If a device has dual Routing Engines and you want the script to be updated on both Routing Engines, you must include the **refresh** or **refresh-from** statements in the configuration of both Routing Engines. The **commit synchronize** command does not cause the **refresh** or **refresh-from** statement to update scripts on both Routing Engines.

CAUTION: For commit scripts, we recommend that you do not automate the update function by including the **refresh** statement as a commit script change element. Even though this might seem like a good way to ensure that the most current commit script is always used, we recommend against it for the following reasons:

- Automated update means that the network must be operational for the commit operation to succeed. If the network goes down after you make a configuration error, you cannot recover quickly.
- If multiple commit scripts need to be updated during each commit operation, the network response time can slow down.
- Automated update is always the last action performed during a commit operation. Consequently, the updated commit script executes only during the next commit operation. This is because commit scripts are applied to the candidate configuration before the software copies any persistent changes generated by the scripts to the candidate configuration. In contrast, if you perform the update operation manually, the updated commit script takes effect as expected, that is, immediately after you commit the **refresh** statement in the configuration.
- If you automate the update operation, the **refresh-from** statement has no effect, because the **refresh-from** URL conflicts with and is overridden by the **source** statement URL. For information about the **refresh-from** statement, see "Using an Alternate Source Location for a Script" on page 222.

## Using a Master Source Location for a Script



### Configuring and Refreshing from the Master Source for a Script

You can store a master copy of each script in a central repository. This eases file management because you can make changes to the master script in one place and then update the copy on each device where the script is currently enabled. This section discusses the following concepts:



#### *Configuring the Master Source for a Script*

To specify the location of the master source for a script, configure the **source** statement and the URL of the master file at the hierarchy level where the script is configured. Including the **source** statement in the configuration does not affect the local copy of the script until you issue the **set refresh** command. At that point, the device retrieves the master copy from the specified URL and overwrites the local copy.

The hierarchy location for the **source** statement depends on the script type and filename.

**commit script**—Include the **source** statement at the **[edit system scripts commit file *filename*]** hierarchy level.

```
[edit system scripts commit file filename]
user@R1# set source url
```

**op script**—Include the **source** statement at the **[edit system scripts op file *filename*]** hierarchy level.

```
[edit system scripts op file filename]
user@R1# set source url
```

**event script**—Include the **source** statement at the **[edit event-options event-script file *filename*]** hierarchy level.

```
[edit event-options event-script file filename]
user@R1# set source url
```

Where

- ***filename***—Name of the script.
- ***url***—URL of the script's master source file. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

#### *Updating a Script from the Master Source*

If you configure a master source for one or more scripts on a device, you can refresh the scripts on that device using the **set refresh** configuration mode command. You can update

a single script or all scripts of a given script type that have a master source location configured.

The update operation occurs as soon as you issue the **set refresh** command. When you issue the **set refresh** command, the switch, router, or security device immediately attempts to connect to the specified URL and retrieve a copy of the master file. The master copy overwrites the local script stored in the scripts directory on the device. If a master source is not defined for a script, that script is not updated and a warning is issued. For commit scripts, the updated commit script is executed when you next issue the **commit** command.

Issuing the **set refresh** command does not add the **refresh** statement to the configuration. Thus the command behaves like an operational mode command by executing an operation, instead of adding a statement to the configuration.

If the device has dual Routing Engines and you want to update a script on both Routing Engines, you must issue the **set refresh** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh** statement to update scripts on both Routing Engines.

To update a single script from its master source, issue the **set refresh** command at the hierarchy level where the script is configured. The **source** statement specifying the master source location must already be configured.

**commit script**—Issue the **set refresh** command at the **[edit system scripts commit file *filename*]** hierarchy level.

```
[edit system scripts commit file filename]
user@R1# set refresh
```

**op script**—Issue the **set refresh** command at the **[edit system scripts op file *filename*]** hierarchy level.

```
[edit system scripts op file filename]
user@R1# set refresh
```

**event script**—Issue the **set refresh** command at the **[edit event-options event-script file *filename*]** hierarchy level.

```
[edit event-options event-script file filename]
user@R1# set refresh
```

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

To update all enabled scripts of a given script type from their master source files, issue the **set refresh** command at the hierarchy level for that script type.

**commit scripts**—Issue the **set refresh** command at the **[edit system scripts commit]** hierarchy level:

```
[edit system scripts commit]
user@R1# set refresh
```

**op scripts**—Issue the **set refresh** command at the **[edit system scripts op]** hierarchy level:

```
[edit system scripts op]
```

```
user@R1# set refresh
```

**event scripts**—Issue the **set refresh** command at the **[edit event-options event-script]** hierarchy level:

```
[edit event-options event-script]
user@R1# set refresh
```

## Example: Configuring and Refreshing from the Master Source for a Script

The following example configures a master source file for an op script on a device running Junos OS. The remote source is defined as an HTTP URL. The example uses the master source to update the local copy of the script on the device.



### *Requirements*

- Routing, switching, or security device running Junos OS.

### *Overview*

You can store a master copy of each script in a central repository. You can make changes to the master script in one place and then update the local copy of the script on devices where the script is enabled.

This example enables the op script **iso.xsl** on a device running Junos OS and then configures a master source location for the script. The remote source for the **iso.xsl** file is the HTTP URL http://my.example.com/pub/scripts/iso.xsl.

Once you configure the master source location, you refresh the local script by issuing the **set refresh** configuration mode command at the hierarchy level where you configured the script. In this example, you would issue the **set refresh** command at the **[edit system scripts op file iso.xsl]** hierarchy level.

### *Configuration*

**Step-by-Step Procedure**

To download, enable, and configure the master source location for the script:

1. Copy the script to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, configure the **file** statement to enable the **iso.xsl** script.

   ```
   [edit system scripts op]
   user@R1# set file iso.xsl
   ```

3. To configure the master source for the **iso.xsl** file, include the **source** statement and source location at the **[edit system scripts op file iso.xsl]** hierarchy level.

   ```
   [edit system scripts op file iso.xsl]
   user@R1# set source http://my.example.com/pub/scripts/iso.xsl
   ```

4. Issue the **commit and-quit** command to commit the configuration and exit to operational mode.

```
[edit]
user@R1# commit and-quit
```

**Results**

```
system {
  scripts {
    op {
      file iso.xsl {
        source http://my.example.com/pub/scripts/iso.xsl;
      }
    }
  }
}
```

*Verifying the Script*

**Purpose** Verify that the script is on the device and enabled in the configuration.

**Action** Issue the **file list** operational mode command to view the files in the specified directory. The **detail** option provides additional information such as permissions, file size, and modified date.

```
user@R1> file list /var/db/scripts/op detail

/var/db/scripts/op:
total 128
-rw-r--r--  1 root  admin  13897 Feb 10  2011 iso.xsl
...
```

Issue the **show configuration system scripts op** operational mode command to list the op scripts currently enabled on the device.

```
user@R1> show configuration system scripts op
file iso.xsl
```

*Refreshing the Script from the Master Source*

**Step-by-Step Procedure** To refresh the local copy of the script from the master source file:

1. In configuration mode, issue the **set refresh** command at the **[edit system scripts op file iso.xsl]** hierarchy level.

```
[edit system scripts op file iso.xsl]
user@R1# set refresh
```

*Verification*

*Verifying the Updated Script*

**Purpose** After refreshing the script, verify that the local copy is updated.

**Action** Issue the **file list** operational mode command with the **detail** option to view the files in the specified directory. Verify that the modified date reflects the refreshed version.

```
user@R1> file list /var/db/scripts/op detail

/var/db/scripts/op:
total 128
-rw-r--r--  1 root  admin  14128 May 26  2011 iso.xsl
...
```

**Related Documentation**

- Using an Alternate Source Location for a Script on page 222
- refresh (Commit Scripts) on page 452
- refresh (Op Scripts) on page 544
- refresh (Event Scripts) on page 690

## Using an Alternate Source Location for a Script



### Refreshing a Script from an Alternate Location

In addition to updating a script from the master source defined by the **source** statement, you also can update a script from an alternate location using the **set refresh-from** configuration mode command. This is convenient when, for example, the master source cannot be accessed due to network issues or other problems.

The update operation occurs as soon as you issue the **set refresh-from** command. When you issue the **set refresh-from** command, the switch, router, or security device immediately attempts to connect to the specified URL and retrieve a copy of the file. The copy overwrites the local script stored in the scripts directory on the device. If a copy of the source is not available, that script is not updated and a warning is issued. For commit scripts, the updated commit script is executed when you next issue the **commit** command. You can also refresh a single script from the remote location by issuing the **request system scripts refresh-from** operational mode command and specifying the URL for the remote script.

Issuing the **set refresh-from** command does not add the **refresh-from** statement to the configuration. Thus the **set refresh-from** command behaves like an operational mode command by executing an operation, instead of adding a statement to the configuration.

If a device has dual Routing Engines and you want the script to be updated on both Routing Engines, you must issue the **set refresh-from** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh-from** statement to update scripts on both Routing Engines.

When you issue the **set refresh-from** command, Junos OS creates a folder in the **/var/tmp** directory. This folder is used for file transfer. After the transfer and refresh operations are complete, Junos OS deletes the temporary folder.

To update a single script from the alternate source, issue the **set refresh-from** command at the hierarchy level where the script is configured, and specify the location of the remote file.

**commit script**—Issue the **set refresh-from** command at the **[edit system scripts commit file *filename*]** hierarchy level.

```
[edit system scripts commit file filename]
user@R1# set refresh-from url
```

**op script**—Issue the **set refresh-from** command at the **[edit system scripts op file *filename*]** hierarchy level.

```
[edit system scripts op file filename]
user@R1# set refresh-from url
```

**event script**—Issue the **set refresh-from** command at the **[edit event-options event-script file *filename*]** hierarchy level.

```
[edit event-options event-script file filename]
user@R1# set refresh-from url
```

To update all enabled scripts of a given script type from an alternate source, issue the **set refresh-from** command at the hierarchy level for that script type, and specify the location of the remote directory that houses the scripts.

**commit script**—Issue the **set refresh-from** command at the **[edit system scripts commit]** hierarchy level.

```
[edit system scripts commit]
user@R1# set refresh-from url
```

**op script**—Issue the **set refresh-from** command at the **[edit system scripts op]** hierarchy level.

```
[edit system scripts op]
user@R1# set refresh-from url
```

**event script**—Issue the **set refresh-from** command at the **[edit event-options event-script]** hierarchy level.

```
[edit event-options event-script]
user@R1# set refresh-from url
```

Where

***url***—URL of the remote script or directory. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

### Example: Refreshing a Script from an Alternate Source

The following example uses an alternate source location to update the local copy of the script on a device running Junos OS. The remote source is defined as an HTTP URL.


- Requirements on page 224
- Overview on page 224

*Requirements*

- Routing, switching, or security device running Junos OS.

*Overview*

You can update a script from a location other than that of the master source. This is convenient when, for example, the master source cannot be accessed due to network issues or other problems. You can refresh a single script or all scripts of a given type from the alternate location.

This example enables the op script **iso.xsl** on a device running Junos OS and then refreshes the script from a location other than the master source location. The remote source for the **iso.xsl** file is the HTTP URL http://my.example.com/pub/scripts2/iso.xsl.

You refresh the local script by issuing the **set refresh-from** configuration mode command at the hierarchy level where you configured the script. In this example, you would issue the **set refresh-from** command at the **[edit system scripts op file iso.xsl]** hierarchy level.

*Configuration*

**Step-by-Step Procedure**

To download and enable the script:

1. Copy the script to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, configure the **file** statement to enable the **iso.xsl** script.

   ```
   [edit system scripts op]
   user@R1# set file iso.xsl
   ```

3. Issue the **commit and-quit** command to commit the configuration and exit to operational mode.

   ```
   [edit]
   user@R1# commit and-quit
   ```

**Results**

```
system {
  scripts {
    op {
      file iso.xsl;
    }
  }
}
```

*Verifying the Script*

**Purpose** Verify that the script is on the device and enabled in the configuration.

**Action** Issue the **file list** operational mode command to view the files in the specified directory. The **detail** option provides additional information such as permissions, file size, and modified date.

```
user@R1> file list /var/db/scripts/op detail

/var/db/scripts/op:
total 128
-rw-r--r--  1 root  admin  13897 Feb 10  2011 iso.xsl
...
```

Issue the **show configuration system scripts op** operational mode command to list the op scripts currently enabled on the device.

```
user@R1> show configuration system scripts op
file iso.xsl
```

*Refreshing the Script from the Alternate Location*

**Step-by-Step Procedure** To refresh the local copy of the script from the alternate location:

1. In configuration mode, issue the **set refresh-from** command at the **[edit system scripts op file iso.xsl]** hierarchy level.

   ```
   [edit system scripts op file iso.xsl]
   user@R1# set refresh-from http://my.example.com/pub/scripts2/iso.xsl
   ```

*Verification*

*Verifying the Updated Script*

**Purpose** After refreshing the script, verify that the local copy is updated.

**Action** Issue the **file list** operational mode command with the **detail** option to view the files in the specified directory. Verify that the modified date reflects the refreshed version.

```
user@R1> file list /var/db/scripts/op detail

/var/db/scripts/op:
total 128
-rw-r--r--  1 root  admin  14128 May 26  2011 iso.xsl
...
```

**Related Documentation**

- Using a Master Source Location for a Script on page 218
- refresh-from (Commit Scripts) on page 453
- refresh-from (Op Scripts) on page 545
- refresh-from (Event Scripts) on page 690

CHAPTER 11

# Configuring Limits on Concurrent Event Policies and Memory Allocation for Scripts



## Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts



### Overview of Limits on Executed Event Policies and Memory Allocation for Scripts

By default, the maximum number of event policies that can run concurrently in the system is 15, and the maximum amount of memory allocated for the data segment portion of an executed script is half of the total available memory of the system, up to a maximum value of 128 MB. If a script requires more memory during execution than the set maximum limit, the script exits. If the system is running the maximum number of event policies, the system ignores any triggered event policy until such time that another policy finishes. The system logs the EVENTD_POLICY_LIMIT_EXCEEDED message for any triggered event policies that were not executed.

Starting with Junos OS Release 12.3, you can configure limits on the maximum number of concurrently running event policies and the maximum amount of memory allocated for the data segment for scripts of a given type.

Depending on the device and its function in the network, it might be necessary to configure larger or smaller limits on the number of event policies that can execute concurrently and the maximum amount of memory allocated to scripts. You might configure smaller limits on critical devices to ensure that priority processes are not adversely impacted, and that the device can perform all necessary functions in the network. Additionally, during normal device operation, you might want to allocate disproportionate amounts of memory to different script types. For example, a device might have a particular type of script that plays a vital role in its operation and requires a specific amount of memory to ensure proper execution.

To set the maximum number of event policies that can run concurrently on a device, configure the **max-policies *policies*** statement at the **[edit event-options]** hierarchy level. You can configure a maximum of 0 through 20 policies.

```
[edit]
event-options {
   max-policies policies;
}
```

To set the maximum memory allocated to the data segment for scripts of a given type, configure the **max-datasize *size*** statement under the hierarchy appropriate for that script type, where ***size*** is the memory in bytes. To specify the memory in kilobytes, megabytes, or gigabytes, append **k**, **m**, or **g**, respectively, to the size. You can configure the memory in the range from 2,3068,672 bytes (22 MB) through 1,073,741,824 bytes (1 GB).

```
[edit]
event-options {
   event-script {
      max-datasize size;
   }
}
system {
   scripts {
      commit {
         max-datasize size;
      }
      op {
         max-datasize size;
      }
   }
}
```

When the **max-datasize** statement is configured and a script executes, Junos OS sets the maximum memory limit for that script to the configured value irrespective of the total memory available on the system at the time of execution. If the script exceeds the maximum memory limit during execution, it exits gracefully.

## Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts

This example configures a maximum value for the number of event polices that the device can execute concurrently and a maximum memory limit for executed commit, event, and op scripts.



### Requirements

A device running Junos OS Release 12.3 or later.

## Overview

This example configures a device running Junos OS to limit the number of event policies that can run simultaneously on that device to a maximum of 12 policies. Additionally, the example configures each script type with a maximum amount of memory that the system can allocate to the data segment portion of a script of that type. The device is configured to allocate 192 MB for each executed commit script and event script and 100 MB for each executed op script.

## Configuration

**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set system scripts commit max-datasize 192m
set system scripts op max-datasize 100m
set event-options max-policies 12
set event-options event-script max-datasize 192m
```

**Step-by-Step Procedure**

1. Configure the maximum number of event policies that can execute concurrently.

   ```
   [edit]
   user@host# set event-options max-policies 12
   ```

2. Configure the maximum memory allocated for the data segment for each executed commit script.

   ```
   [edit]
   user@host# set system scripts commit max-datasize 192m
   ```

3. Configure the maximum memory allocated for the data segment for each executed op script.

   ```
   [edit]
   user@host# set system scripts op max-datasize 100m
   ```

4. Configure the maximum memory allocated for the data segment for each executed event script.

   ```
   [edit]
   user@host# set event-options event-script max-datasize 192m
   ```

5. Commit the configuration.

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

***Results***

```
[edit]
event-options {
   event-script {
      max-datasize 192m;
   }
```

```
    max-policies 12;
  }
  system {
    scripts {
      commit {
        max-datasize 192m;
      }
      op {
        max-datasize 100m;
      }
    }
  }
```

## Verification

Confirm that the configuration is working properly.

***Verifying the Limit on Concurrently Executing Event Policies***

**Purpose** If the system is running the maximum number of event policies, the system ignores any triggered event policy until such time that another policy finishes. The system logs the EVENTD_POLICY_LIMIT_EXCEEDED messsage for any triggered event policies that were not executed. By default, system log messages are recorded in the **messages** log file.

**Action** Review the configured log file to verify whether any policies were barred from execution, because the maximum limit was reached. You can narrow the output to include only the relevant error messages by appending **| match EVENTD_POLICY_LIMIT_EXCEEDED**.

```
user@R1> show log messages | match EVENTD_POLICY_LIMIT_EXCEEDED
Jun 11 17:02:42  R1 eventd[1177]: EVENTD_POLICY_LIMIT_EXCEEDED: Unable to execute
 policy 'raise-trap' because current number of policies (12) exceeds system limit
 (12)
[output omitted]
```

**Related Documentation**

- max-datasize on page 450
- max-policies on page 640

CHAPTER 12

# Specifying the Session Protocol in Scripts



## Specifying the Session Protocol for Connections Using Junos Automation Scripts



### Session Protocol in Junos Automation Scripts Overview

The Junos XML management protocol is a Juniper Networks proprietary protocol used to request information from and configure devices running Junos OS. The NETCONF XML management protocol is a standard used to request and change configuration information on a routing, switching, or security device. The NETCONF protocol is defined in RFC 4741, *NETCONF Configuration Protocol*, which is available at http://www.ietf.org/rfc/rfc4741.txt .

Starting with Junos OS Release 11.4, the **jcs:open( )** function includes the option to create a session either with the Junos XML protocol server on devices running Junos OS or with the NETCONF server on devices where NETCONF service over SSH is enabled. Previously, the function supported only sessions with the Junos XML protocol server on devices running Junos OS. The additional support for NETCONF sessions permits automation scripts to configure and manage devices in a multi-vendor environment.

The **jcs:open( )** function supports the following session protocol types:

- **junoscript**—Session with the Junos XML protocol server on a routing, switching, or security device running Junos OS. This session type supports the operations defined in the Junos XML protocol and the Junos XML API, which are used to configure devices running Junos OS or to request information about the device configuration or operation. This is the default session type.
- **netconf**—Session with the NETCONF XML protocol server on a routing, switching, or security device over an SSHv2 connection. The device to which the connection is made must be enabled for NETCONF service over SSH. NETCONF over SSH is described in RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, which is available at http://www.ietf.org/rfc/rfc4742.txt .

- **junos-netconf**—Proprietary session with the NETCONF XML protocol server over an SSHv2 connection on a routing, switching, or security device running Junos OS.

The NETCONF server on a device running Junos OS has the additional capabilities defined in **http://xml.juniper.net/netconf/junos/1.0** . The NETCONF server on these devices supports NETCONF XML protocol operations, most Junos XML protocol operations, and the tag elements defined in the Junos XML API. For **netconf** and **junos-netconf** sessions with devices running Junos OS, you should use only native NETCONF XML protocol operations and the extensions available in the Junos XML protocol for configuration functions as documented in the *NETCONF XML Management Protocol Guide*.

The syntax for the **jcs:open()** function when specifying a session protocol is:

**SLAX Syntax**

```
var $connection = jcs:open(remote-hostname, session-options);
```

**XSLT Syntax**

```
<xsl:variable name="connection" select="jcs:open(remote-hostname,session-options)"/>
```

The *session-options* parameter is an XML node-set that specifies the session type and connection parameters. The session type is one of three values: **junoscript**, **netconf**, or **junos-netconf**. If you do not specify a session type, the default is **junoscript**, which opens a session with the Junos XML protocol server only on devices running Junos OS. The format of the node-set is:

```
var $session-options := {
    <method> ("junoscript" | "netconf" | "junos-netconf");
    <username> "username";
    <passphrase> "passphrase";
    <password> "password";
    <port> "port-number";
    <instance> "routing-instance-name";
    <routing-instance> "routing-instance-name";
  }
```

If you do not specify a username and it is required for the connection, the script uses the local name of the user executing the script. The **<passphrase>** and **<password>** elements serve the same purpose. If you do not specify a passphrase or password element and it is required for authentication, you should be prompted for one during script execution by the device to which you are connecting.

Optionally, you can specify the server port number for **netconf** and **junos-netconf** sessions. The default NETCONF server port number is 830. If you do not specify a port number for a **netconf** or **junos-netconf** session, **jcs:open()** connects to the NETCONF server using port 830. However, if you specify a port number, **jcs:open()** connects to the given port instead. Specifying a port number has no impact on **junoscript** sessions, which are always established over SSH port 22.

To redirect the SSH connection to originate from within a specific routing instance, include the **instance** or **routing-instance** element and the routing instance name. The routing instance must be configured at the **[edit routing-instances]** hierarchy level. The remote device must be reachable either using the routing table for that routing instance or from one of the interfaces configured under that routing instance. The **instance** and **routing-instance** elements serve the same purpose.

To verify the protocol for a specific connection, call the **jcs:get-protocol(*connection*)** extension function and pass the connection handle as the argument. The function returns "junoscript", "netconf", or "junos-netconf", depending on the session type.

During session establishment with a NETCONF server, the client application and NETCONF server each emit a **<hello>** tag element to specify which operations, or *capabilities,* they support from among those defined in the NETCONF specification or published as proprietary extensions. In **netconf** and **junos-netconf** sessions, you can retrieve the session capabilities of the NETCONF server by calling the **jcs:get-hello(*connection*)** extension function.

For example, the NETCONF server on a typical device running Junos OS might return the following capabilities:

```
<hello>
  <capabilities>
     <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
     <capability>
       urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
     </capability>
     <capability>
       urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
     </capability>
     <capability>
       urn:ietf:params:xml:ns:netconf:capability:validate:1.0
     </capability>
     <capability>
       urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file
     </capability>
     <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
     <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>20826</session-id>
</hello>
```

## Example: Specifying the Session Protocol for a Connection Using an Automation Script

The following example demonstrates how to specify the session protocol within a Junos automation script when creating a connection with a remote device. Specifically, the example op script establishes a NETCONF session with a remote device running Junos OS, retrieves and prints the NETCONF server capabilities, and then updates and commits the configuration on that device.



### Requirements

- Routing, switching, or security device running Junos OS Release 11.4 or later.
- Client application can log in to the device where the NETCONF server resides.

- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

### Overview and Script

Starting with Junos OS Release 11.4, the **jcs:open( )** function includes the option to create a session either with the Junos XML protocol server on devices running Junos OS or with the NETCONF server on devices where NETCONF service over SSH is enabled. In the following example, the script creates a connection and establishes a NETCONF session with a remote device running Junos OS. If the connection and session are successfully established, the script updates the configuration on the remote device to add the **ftp** statement to the **[edit system services]** hierarchy level. The script also retrieves and prints the session protocol and the capabilities of the NETCONF server.

The script takes one argument, **remote-host**, which is the IP address or hostname of the remote device. The **arguments** variable is declared at the global level of the script so that the argument name and description are visible in the command-line interface (CLI) when a user requires context-sensitive help.

The variable **netconf** is a node-set that specifies the session protocol and the connection parameters for the remote device. The value of the **<method>** element is set to “netconf” to establish a session with the NETCONF server over an SSHv2 connection. The **<username>** element specifies the username for the connection. If you do not specify a username and it is required for the connection, the script uses the local name of the user executing the script. In this example, the passphrase and port are not specified. If a passphrase is required for authentication, the remote device should prompt for one during script execution. The script establishes the session using the default NETCONF port 830.

If the connection and establishment of the NETCONF session are successful, the script executes remote procedure calls (RPCs). The RPCs contain the tag elements **<lock>**, **<edit-config>**, **<commit>**, and **<unlock>**, which are NETCONF operations to lock, edit, commit, and unlock the candidate configuration. The script stores the RPC for each task in a separate variable. The results for each RPC are also stored separately and parsed for errors. The script only executes each subsequent step if the previous step is successful. For example, if the script cannot lock the configuration, it does not execute the RPCs to edit, commit, or unlock the configuration.

The variable **rpc-edit-config** contains the tag element **<edit-config>**, which is a NETCONF operation to modify a configuration. The child element, **<config>**, includes the modified portion of the configuration that is merged with the candidate configuration on the device. If errors are encountered, the script calls the **copy-of** statement to copy the result tree fragment variable to the results tree so that the error message prints to the CLI during script execution.

**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";
ns ext = "http://xmlsoft.org/XSLT/namespace";

var $arguments = {
```

```
  <argument> {
    <name> "remote-host";
    <description> "device hostname or IP address to which to connect";
  }
}
param $remote-host;

match / {

  <op-script-results> {

    var $netconf := {
      <method> "netconf";
      <username> "bsmith";
    }

    var $rpc-lock-config = {
      <lock> {
        <target> {
          <candidate>;
        }
      }
    }

    var $rpc-unlock-config = {
      <unlock> {
        <target> {
          <candidate>;
        }
      }
    }

    var $rpc-commit = {
      <commit>;
    }

    var $rpc-edit-config = {
      <edit-config> {
        <target> {
          <candidate>;
        }
        <default-operation> "merge";
        <config> {
          <configuration> {
            <system> {
              <services> {
                <ftp>;
              }
            }
          }
        }
      }
    }

    if ($remote-host = '') {
      <xnm:error> {
```

```
      <message> "missing mandatory argument 'remote-host'";
    }
  }
  else {

    var $connection = jcs:open($remote-host, $netconf);
    if ($connection) {

      /* request protocol and capabilities */
      var $protocol = jcs:get-protocol($connection);
      var $capabilities = jcs:get-hello($connection);

      <output> "\nSession protocol: " _ $protocol _ "\n";
      copy-of $capabilities;

      /* execute rpcs to lock, edit, commit, and unlock config */
      var $lock-reply = jcs:execute($connection, $rpc-lock-config);
      if ($lock-reply/..//rpc-error) {
        copy-of $lock-reply;
      }
      else {
        var $edit-config-reply = jcs:execute($connection, $rpc-edit-config);
        if ($edit-config-reply/..//rpc-error) {
          <output>"Configuration error: "_ $edit-config-reply/..//error-message/.
                _ "\nConfiguration not committed.\n" ;
          copy-of $edit-config-reply;
        }
        else {
          var $commit-reply = jcs:execute($connection, $rpc-commit);
          if ($commit-reply/..//rpc-error) {
            <output>"Commit error or warning: " _ $commit-reply/..//error-message/.;
            copy-of $commit-reply;
          }
        }
        var $unlock-reply = jcs:execute($connection, $rpc-unlock-config);
      }

      expr jcs:close($connection);
    }
    else {
      <output>"\nNo connection - exiting script";
    }

  }
 }
}
```

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet 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"
xmlns:ext="http://xmlsoft.org/XSLT/namespace" version="1.0">

 <xsl:variable name="arguments">
  <argument>
```

```
  <name>remote-host</name>
  <description>device hostname or IP address to which to connect</description>
 </argument>
</xsl:variable>

<xsl:param name="remote-host"/>

<xsl:template match="/">
 <op-script-results>
  <xsl:variable name="netconf-temp-1">
   <method>netconf</method>
   <username>bsmith</username>
  </xsl:variable>
  <xsl:variable xmlns:ext="http://xmlsoft.org/XSLT/namespace"
   name="netconf" select="ext:node-set($netconf-temp-1)"/>

  <xsl:variable name="rpc-lock-config">
   <lock>
    <target>
     <candidate/>
    </target>
   </lock>
  </xsl:variable>

  <xsl:variable name="rpc-unlock-config">
   <unlock>
    <target>
     <candidate/>
    </target>
   </unlock>
  </xsl:variable>

  <xsl:variable name="rpc-commit">
   <commit/>
  </xsl:variable>

  <xsl:variable name="rpc-edit-config">
   <edit-config>
    <target>
     <candidate/>
    </target>
    <default-operation>merge</default-operation>
    <config>
     <configuration>
      <system>
       <services>
        <ftp/>
       </services>
      </system>
     </configuration>
    </config>
   </edit-config>
  </xsl:variable>

  <xsl:choose>
   <xsl:when test="$remote-host = ''">
```

```
 <xnm:error>
  <message>missing mandatory argument 'remote-host'</message>
 </xnm:error>
</xsl:when>
<xsl:otherwise>
 <xsl:variable name="connection" select="jcs:open($remote-host, $netconf)"/>

 <xsl:choose>
  <xsl:when test="$connection">

   <!-- request protocol and capabilities -->
   <xsl:variable name="protocol" select="jcs:get-protocol($connection)"/>
   <xsl:variable name="capabilities" select="jcs:get-hello($connection)"/>
   <output>
    <xsl:value-of select="concat('&#10;Session protocol: ', $protocol, '&#10;')"/>
   </output>
   <xsl:copy-of select="$capabilities"/>

   <!-- execute rpcs -->
   <xsl:variable name="lock-reply"
    select="jcs:execute($connection, $rpc-lock-config)"/>
   <xsl:choose>
    <xsl:when test="$lock-reply/..//rpc-error">
     <xsl:copy-of select="$lock-reply"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:variable name="edit-config-reply"
      select="jcs:execute($connection, $rpc-edit-config)"/>
     <xsl:choose>
      <xsl:when test="$edit-config-reply/..//rpc-error">
       <output>
        <xsl:value-of select="concat('Configuration error: ',
         $edit-config-reply/..//error-message/.,
         '&#10;Configuration not committed.&#10;')"/>
       </output>
       <xsl:copy-of select="$edit-config-reply"/>
      </xsl:when>
      <xsl:otherwise>
       <xsl:variable name="commit-reply"
        select="jcs:execute($connection, $rpc-commit)"/>
       <xsl:if test="$commit-reply/..//rpc-error">
        <output>
         <xsl:value-of select="concat('Commit error or warning: ',
          $commit-reply/..//error-message/.)"/>
        </output>
        <xsl:copy-of select="$commit-reply"/>
       </xsl:if>
      </xsl:otherwise>
     </xsl:choose>
     <xsl:variable name="unlock-reply" select="jcs:execute($connection,
      $rpc-unlock-config)"/>
    </xsl:otherwise>
   </xsl:choose>

   <xsl:value-of select="jcs:close($connection)"/>
  </xsl:when>
```

```
        <xsl:otherwise>
          <output>No connection - exiting script</output>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
 </op-script-results>
</xsl:template>
</xsl:stylesheet>
```

## 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 **netconf-session.xsl** or **netconf-session.slax** as appropriate, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **netconf-session.xsl** or **netconf-session.slax** as appropriate.

   ```
   [edit system scripts op]
   bsmith@local-host# set file netconf-session.(slax | xsl)
   ```

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

   ```
   [edit]
   bsmith@local-host# commit and-quit
   ```

4. Execute the op script on the local device by issuing the **op netconf-session** operational mode command and include any necessary arguments.

   In this example, the user, bsmith, is connecting to the remote device, fivestar. The remote device has dual routing engines, so the **commit** operation returns a warning that the **commit synchronize** command should be used to commit the new candidate configuration to both routing engines.

   ```
   bsmith@local-host> op netconf-session remote-host fivestar
   bsmith@fivestar's password:
   Session protocol: netconf
   Commit error or warning:
   graceful-switchover is enabled, commit synchronize should be used
   ```

## Verification

Confirm that the device is working properly.


- Verifying Op Script Execution on page 239
- Verifying the Configuration Changes on page 241


### *Verifying Op Script Execution*

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

**Action** Review the script output in the CLI and in the op script log file. Take particular note of any errors that occurred during execution. The default op script log file is

**/var/log/op-script.log**. If the log file is significantly lengthy, limit the display by appending the **| last *number-of-lines*** option to the **show log** command and specify the number of lines to print to the CLI. The output within the **<op-script-results>** element is relevant to the script execution.

```
bsmith@local-host> show log op-script.log | last 100
...output omitted for brevity...
<op-script-results xmlns:junos="http://xml.juniper.net/junos/*/junos"
xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm" xml
ns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0"
xmlns:ext="http://xmlsoft.org/XSLT/namespace">
<output>
Session protocol: netconf
</output>
<hello>
  <capabilities>
   <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
   <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
   </capability>
   <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
   </capability>
   <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>

   <capability>
      urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file
   </capability>
   <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
   <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>29087</session-id>
</hello>
  <output>Commit error or warning:
graceful-switchover is enabled, commit synchronize should be used
</output>
  <rpc-error>
<error-severity>warning</error-severity>
<error-message>
graceful-switchover is enabled, commit synchronize should be used
</error-message>
</rpc-error>
  <ok/>
</op-script-results>
```

You can also obtain more descriptive script output on a device running Junos OS by including the **| display xml** option when you execute an op script.

```
bsmith@local-host> op netconf-session remote-host fivestar | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4D0/junos">
  <output>
      Session protocol: netconf
  </output>
  <hello>
   <capabilities>
     <capability>
       urn:ietf:params:xml:ns:netconf:base:1.0
     </capability>
     <capability>
       urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
     </capability>
     <capability>
```

```
      urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
    </capability>
    <capability>
      urn:ietf:params:xml:ns:netconf:capability:validate:1.0
    </capability>
    <capability>
      urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file
    </capability>
    <capability>
      http://xml.juniper.net/netconf/junos/1.0
    </capability>
    <capability>
      http://xml.juniper.net/dmi/system/1.0
    </capability>
   </capabilities>
   <session-id>
       29087
   </session-id>
  </hello>
  <output>
      Commit error or warning:
      graceful-switchover is enabled, commit synchronize should be used
  </output>
  <rpc-error>
      <error-severity>
          warning
      </error-severity>
      <error-message>
          graceful-switchover is enabled, commit synchronize should be used
      </error-message>
  </rpc-error>
  <ok/>
</op-script-results>
<cli>
  <banner></banner>
</cli>
</rpc-reply>
```

**Meaning** This example creates a NETCONF session on a remote device running Junos OS. The capabilities of the NETCONF server include both standard NETCONF operations and Juniper Networks proprietary extensions, which are defined in **http://xml.juniper.net/netconf/junos/1.0** and **http://xml.juniper.net/dmi/system/1.0** . The RPC results for the **commit** operation include one warning, but the commit operation is still successful.

### *Verifying the Configuration Changes*

**Purpose** Verify that the commit was successful by viewing the configuration change and the commit log on the remote device.

**Action** On the remote device, execute the **show configuration system services** operational mode command to view the **[edit system services]** hierarchy level of the configuration. If the script is successful, the configuration includes the **ftp** statement.

```
bsmith@fivestar> show configuration system services
ftp;
netconf {
    ssh;
}
```

Additionally, you can review the commit log. On the remote device, execute the **show system commit** operational mode command to view the commit log. In this example, the log confirms that bsmith committed the candidate configuration in a NETCONF session at the given date and time.

```
bsmith@fivestar> show system commit
0   2011-07-11 12:04:01 PDT by bsmith via netconf
1   2011-07-08 15:16:33 PDT by root via cli
```

## Troubleshooting



### *Troubleshooting Connection Errors*

**Problem** The script generates the following error message:

```
hello packet:1:(0) Document is empty
hello packet:1:(0) Start tag expected, '<' not found
error: netconf: could not read hello
error: did not receive hello packet from server
error: Error in creating the session with "fivestar" server
No connection - exiting script
```

Potential causes for the connection error include:

- The device or interface to which you are connecting is down or unavailable.
- The script argument for the IP address or DNS name of the remote device is incorrect.
- The connection timeout value was exceeded before establishing the connection.
- The user authentication for the remote device is not valid or is entered incorrectly.
- You are trying to establish a NETCONF session, and NETCONF over SSH is not enabled on the device where the NETCONF server resides, or it is enabled on a different port.

**Solution** Ensure that the remote device is up and running and that the user has access to the device. Also verify that you supplied the correct argument for the IP address or DNS name of the remote device when executing the script.

For NETCONF sessions, ensure that you have enabled NETCONF over SSH on the device where the NETCONF server resides. Since the example program does not specify a specific port number for the NETCONF session, the session is established on the default

NETCONF-over-SSH port, 830. To verify whether NETCONF over SSH is enabled on the default port for a device running Junos OS, enter the following operational mode command on the remote device:

```
bsmith@fivestar> show configuration system services

netconf {
    ssh;
}
```

If the **netconf** configuration hierarchy is absent on the remote device, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

```
[edit]
bsmith@fivestar# set system services netconf ssh
bsmith@fivestar# commit
```

If the **netconf** configuration hierarchy specifies a port other than the default port, include the port number in the XML node-set that you pass to the **jcs:open( )** function. For example, the following device is configured for NETCONF over SSH on port 12345:

```
bsmith@fivestar> show configuration system services
netconf {
    ssh {
       port 12345;
    }
}
```

To create a NETCONF session on the alternate port, include the new port number in the XML node-set.

```
var $netconf := {
  <method> "netconf";
  <username> "bsmith";
  <port> "12345";
}
var $connection = jcs:open($remote-host, $netconf);
...
```

### *Troubleshooting Configuration Lock Errors*

**Problem** The script generates one of the following error messages:

```
configuration database locked by:
  root terminal p0 (pid 24113) on since 2011-07-11 11:48:06 PDT, idle 00:07:59

Users currently editing the configuration:
  root terminal p1 (pid 24279) on since 2011-07-11 12:28:30 PDT
      {master}[edit]

configuration database modified
```

**Solution** Another user currently has a lock on the candidate configuration or has modified the candidate configuration but has not yet committed the configuration. Wait until the lock is released, and then execute the program.

*Troubleshooting Configuration Syntax Errors*

**Problem** The following error message prints to the CLI:

```
Configuration error: syntax error
Configuration not committed.
```

Examine the result tree for additional information. In this case, the result tree shows the following error message:

```
<rpc-error>
   <error-severity>
      error
   </error-severity>
   <error-info>
      <bad-element>
         ftp2
      </bad-element>
   </error-info>
   <error-message>
      syntax error
   </error-message>
</rpc-error>
```

**Solution** The **<bad-element>** tag element indicates that the configuration statement is not valid. Correct the configuration hierarchy and run the script. In this example error, the user entered the tag **<ftp2>** instead of **<ftp>**. Since that is not an acceptable element in the configuration, the NETCONF server returns an error.

**Related Documentation**

- Session Protocol in Junos Automation Scripts Overview on page 231
- *Junos XML Management Protocol Guide*
- *NETCONF XML Management Protocol Guide*
- get-hello( ) Function (jcs Namespace) on page 108
- get-protocol( ) Function (jcs Namespace) on page 110
- open( ) Function (jcs Namespace) on page 112

CHAPTER 13

# Provisioning Services Using Service Template Automation



## Example: Service Template Automation



### Service Template Automation Overview

Starting with Junos OS Release 12.3, you can use service template automation (STA) to provision services such as VPLS VLAN, Layer 2 and Layer 3 VPNs, and IPsec across similar platforms running Junos OS. Service template automation uses the **service-builder.slax** op script to transform a user-defined service template definition into a uniform API, which you can then use to configure and provision services on similar platforms running Junos OS. This permits you to create a service template on one device, generalize the parameters, and then quickly and uniformly provision that service on other devices. This decreases the time required to configure the same service on multiple devices, and reduces configuration errors associated with manually configuring each device.

The following process outlines how to use service template automation to provision services:

1. Create a service template definition.
2. Execute the **service-builder.slax** script and define service-specific instance parameters.
3. Generate the service interface, which automatically builds the required interface (API) from the template.
4. Enable the service interface on each device where the service is required.
5. Provision systems by invoking the service interface using NETCONF and supplying the service parameter values.

You create a new service template by configuring the hierarchies for the actual service to be provisioned on a device running Junos OS. Service template hierarchies are configured at the **[edit groups]** hierarchy level. When creating the service template:

- Do not include **apply-groups** or **apply-macro** statements.
- Do not include any statements that are supported on the current device that are not also supported on the devices where the service will be provisioned (for example dual Routing Engine versus single Routing Engine).
- Commit the configuration. The service template group configuration is read from the committed configuration.

Once you create the basic service template definition, you invoke the **service-builder.slax** op script. The script reads the service template information from the committed configuration and uses an interactive interface to help you build and generate the service API. You have the option to parameterize every variable in the service template or only selected variables. For each selected variable, you create a generic service template parameter. The **service-builder.slax** script guides you through the creation and configuration of each parameter.

After you define the service template parameters, you generate the service interface. This creates a platform-specific service op script. If the **load-scripts-from-flash** statement is configured, the generated service script is stored in the **/config/scripts/op** directory in flash memory. Otherwise, the generated script is stored in the **/var/db/scripts/op** directory on the hard disk.

To enable the service interface on a device, you enable the generated service script in the configuration as you would any op script. You can enable the service interface on the local device using the **service-builder.slax** script or by manually updating the configuration. To enable the service interface on a similar platform, you must copy the generated service script to the corresponding directory on the new device and enable the service script in the configuration.

To provision the service on a device, invoke the service interface using NETCONF, and supply the necessary values for each parameter. Alternatively, you can invoke the service interface in the CLI by executing the service script and supplying the necessary values for each parameter as command-line arguments to the script. You can direct the service script to create a new service configuration, or update or delete an existing service configuration. The service script makes the changes to the candidate configuration and then commits the configuration. The service script does not support the context-sensitive help and auto-completion features available in the Junos OS CLI.

## Example: Configuring Service Template Automation

This example shows how to use service template automation to provision services across similar platforms running Junos OS.

## Requirements

- Two MX Series devices running Junos OS Release 12.3 or later.

## Overview

This example uses service template automation to provision services on an MX Series router. To use the service template automation **service-builder.slax** script, you must first copy the script to the **/var/db/scripts/op** or **/config/scripts/op** directory and enable the script on the device.

The following process outlines how to use service template automation to provision services:

1. Create a service template definition.
2. Execute the **service-builder.slax** script, and define service-specific instance parameters.
3. Generate the service interface.
4. Enable the service interface on each device where the service is required.
5. Provision systems by invoking the service interface using NETCONF and supplying the service parameter values.

This example creates a new VPN service interface on an MX Series device running Junos OS Release 12.3 and provisions the service on a second MX Series device running Junos OS Release 12.3. You configure service template definitions under the **[edit groups]** hierarchy level. For this example, the service name is **vpn-service**, and the template group name is **vpn-service-template-group**. The **load merge terminal** configuration mode command loads the service template configuration hierarchies into the candidate configuration, which is then committed.

Once you create the initial service template, you execute the **service-builder.slax** script. The script prompts for the service name and the template group name, and then reads the service template configuration from the committed configuration.

The **service-builder.slax** script interface consists of two menus: **Main Menu** and **Hierarchies Menu**. Within the **Main Menu**, you can review the variables defined in the service template configuration, or you can build or enable the service API. The **Build Service API** menu option displays the **Hierarchies Menu**, which steps you through the parameterization of the variables. The default is to parameterize every variable, or you can choose to parameterize selected variables. If you must exit the **service-builder.slax** script while building the service API, you must finish configuring all the parameters for the current hierarchy in order to save that hierarchy configuration when you exit using the **Quit** option. Then you can finish configuring any incomplete hierarchies at a later time. This example parameterizes two variables: the interface name and the interface description. After the parameters are specified, the service builder script generates the service script.

The **Enable Service API** menu option enables the service script on the local device. To enable the service script on the second MX Series device, the generated service script is copied to the **/var/db/scripts/op** directory on the second device, and the script is enabled in the configuration. If the **load-scripts-from-flash** statement is configured, the script must be copied to the corresponding directory on the flash drive instead.

NETCONF is used to provision the service on the remote MX Series device. The NETCONF remote procedure call (RPC) action depends on whether the service is a new service or an existing service. Supported actions include **create**, **update**, and **delete**. This example creates a new service. If the given service is already provisioned on the device and you are updating or deleting the service parameters, you can alter the RPC to perform these actions.

## Configuration



### *Storing and Enabling the Service Builder Script*

**Step-by-Step Procedure**

The Junos OS installation includes the **service-builder.slax** script, which is stored in the **/usr/libexec/scripts/op/** directory on the device. To use the **service-builder.slax** script, you must first copy it to the op scripts directory and enable it in the configuration. Only users in the Junos OS superuser login class can access and edit files in these directories.

1. Copy the **service-builder.slax** script to the **/var/db/scripts/op** directory on the hard disk or the **/config/scripts/op** directory on the flash drive.

   ```
   user@host> file copy /usr/libexec/scripts/op/service-builder.slax /var/db/scripts/op
   ```

2. Verify that the script is in the correct directory by using the **file list** operational mode command.

   ```
   user@host> file list /var/db/scripts/op
   /var/db/scripts/op:
   service-builder.slax*
   ```

3. Enable the script in the configuration.

   ```
   [edit]
   user@host# set system scripts op file service-builder.slax
   ```

4. If you store scripts in and load them from flash, configure the **load-scripts-from-flash** statement, if it is not already configured.

   ```
   [edit]
   user@host# set system scripts load-scripts-from-flash
   ```

5. Commit the configuration.

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

*Configuring the Service Template Definition*

**Step-by-Step Procedure**

To create a new service template on a device running Junos OS:

1. Select a service name.

   This example uses **vpn-service**.

2. In configuration mode, create a new group, which will contain the hierarchies for the actual service to be provisioned.

```
[edit]
user@host# set groups vpn-service-template-group
```

3. Configure the hierarchies for the service.

   For this example, the pre-constructed configuration hierarchies are loaded into the candidate configuration using the **load merge terminal** command.

```
[edit]
user@host# load merge terminal
groups {
  vpn-service-template-group {
    interfaces {
      ge-2/2/6 {
        description "connected to customer3-site-1";
        unit 0 {
          family bridge {
            interface-mode access;
            vlan-id 300;
          }
        }
      }
    }
    protocols {
      rstp {
        interface ge-2/3/0;
      }
      mvrp {
        interface ge-2/3/0;
      }
    }
    bridge-domains {
      bd {
        vlan-id-list 100;
      }
    }
  }
}
[Ctrl+D]
```

4. Verify that the configuration syntax is correct.

```
[edit]
user@host# commit check
configuration check succeeds
```

5. Commit the configuration.

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

*Configuring and Generating the Service Interface*

**Step-by-Step Procedure**

To configure and generate the service interface:

1. In operational mode, execute the **service-builder.slax** script, which starts an interactive Service Builder session.

```
user@host> op service-builder
Welcome to Service Builder Script: (v1.0)
-
Enter the service name :
```

2. Enter the service name that was defined in "Configuring the Service Template Definition" on page 249.

```
Enter the service name : vpn-service
```

3. Enter the group name under which the service hierarchies are configured.

   This example uses the group name **vpn-service-template-group**. The script reads the configuration specified in the **vpn-service-template-group** hierarchy and then displays the main menu.

```
Enter the group name : vpn-service-template-group
.. reading [edit group vpn-service-template-group] ..

[Op Script Builder - Main Menu]
-----------------------------------
 1. Show Variables
 2. Build Service API
 3. Enable Service API
 Q. Quit
-----------------------------------
Enter Selection:>
```

4. (Optional) To review the service template variables that you can parameterize, select the **Show Variables** option.

   The script translates the template definition in the candidate configuration into a general parameter list grouped by hierarchy level.

```
[Op Script Builder - Main Menu]
-----------------------------------
 1. Show Variables
 2. Build Service API
 3. Enable Service API
```

```
 Q. Quit
-----------------------------------
Enter Selection:> 1
 List of variables under each hierarchy to parameterize:
-
 1. [ edit groups vpn-service-template-group interfaces ]
  1.1. interface/name
  1.2. interface/description
  1.3. interface/unit/name
  1.4. interface/unit/family/bridge/interface-mode
  1.5. interface/unit/family/bridge/vlan-id
-
 2. [ edit groups vpn-service-template-group protocols ]
  2.1. rstp/interface/name
  2.2. mvrp/interface/name
-
 3. [ edit groups vpn-service-template-group bridge-domains ]
  3.1. domain/name
  3.2. domain/vlan-id-list
-
```

5. To build the Service API, select the **Build Service API** option.

```
[Op Script Builder - Main Menu]
-----------------------------------
 1. Show Variables
 2. Build Service API
 3. Enable Service API
 Q. Quit
-----------------------------------
Enter Selection:> 2
```

6. From the **Hierarchies Menu**, enter the menu selections for the hierarchies that have variables you want to parameterize, or press Enter to select all hierarchies.

```
[Op Script Builder - Hierarchies Menu]
-----------------------------------
 1. interfaces
 2. protocols
 3. bridge-domains
 Q. Quit
-----------------------------------
Please enter multiple selections separated by a comma (,) only.
-----------------------------------
Enter Selection:> [default:all] 1
```

7. From the variables list, enter the menu selections for the variables you want to parameterize for the service interface, or press Enter to parameterize all variables within that hierarchy.

```
List of variables to parameterize: ...
[ edit groups vpn-service-template-group interfaces ]
-----------------------------------
 1. interfaces/interface/name
 2. interfaces/interface/description
 3. interfaces/interface/unit/name
 4. interfaces/interface/unit/family/bridge/interface-mode
 5. interfaces/interface/unit/family/bridge/vlan-id
```

```
 Q. Quit
Please enter multiple selections separated by a comma (,) only.
---------------------------------
Enter Selection:> [default:all] 1,2
```

8. Configure the selected parameters.

   The system prompts for the required information. This example configures the interface name parameter as **ifname** and the interface description parameter as **ifdesc**.

```
Enter parameter name for: 1.interfaces/interface/name
**************************
[ edit groups vpn-service-template-group interfaces ]
Name for this parameter? ifname
Do you want to revise 'ifname'? (yes/no)[no]: no
Enter parameter name for: 2.interfaces/interface/description
**************************
[ edit groups vpn-service-template-group interfaces ]
Name for this parameter? ifdesc
Do you want to revise 'ifdesc'? (yes/no)[no]: no
```

9. Configure the selected parameters at each hierarchy level.

   The script iterates over each selected hierarchy and the specified parameters. If you must exit the **service-builder.slax** script while building the service API, you must finish configuring all the parameters for the current hierarchy in order to save that hierarchy configuration when you exit using the **Quit** option.

10. Generate the service interface, which creates the service script.

    Once all parameters are configured, the script automatically prompts you to generate the service interface. Press Enter or type yes to generate the service interface.

```
Do you want to commit the previously selected options to create vpn-service
 script?  (yes/no)[yes]: yes
Created service script: /var/db/scripts/op/vpn-service.slax
```

### *Verifying the Service Interface*

**Purpose** Verify the creation of the service script. If the **load-scripts-from-flash** statement is configured, the generated file is stored in flash memory. Otherwise, the generated file is stored on the hard disk.

**Action** Issue the **file list** operational mode command. For this example, the **vpn-service.slax** script should be present in the **/var/db/scripts/op** directory. The **service-builder.slax** script also generates the **utility.slax** script in the **/var/db/scripts/op** directory and the **vpn-service-builder-info.xml** file in the **/var/db/scripts/lib** directory. These files are used by the **service-builder.slax** script and should not be deleted.

```
user@host> file list /var/db/scripts/op
/var/db/scripts/op:
service-builder.slax
utility.slax
vpn-service.slax

user@host> file list /var/db/scripts/lib
/var/db/scripts/lib:
vpn-service-builder-info.xml
```

### *Enabling the Service Interface*

**Step-by-Step Procedure** To enable the service interface on a remote device:

1. Copy the generated service script to the device where you are provisioning the new service.

   If the **load-scripts-from-flash** statement is not configured, copy the service script to the **/var/db/scripts/op** directory on the second device. Otherwise, the script must be copied to the corresponding directory on the flash drive instead.

   Copy the service script to the **/var/db/scripts/op** directory on the second device.

2. Enable the op script in the configuration.

   ```
   [edit]
   user@host2# set system scripts op file vpn-service.slax
   ```

3. Commit the configuration.

   ```
   [edit]
   user@host2# commit
   commit complete
   ```

4. In operational mode, verify that the script is enabled and that the service parameters display as arguments for the script.

   ```
   user@host2> op vpn-service ?
   Possible completions:
    <[Enter]>   Execute this command
    <name>      Argument name
    action      Please enter either create/delete/update
    detail      Display detailed output
    ifdesc      Text description of interface
    ifname      Name of interface
    service-id  Service Name
    |           Pipe through a command
   ```

*Provisioning the Service Using NETCONF*

**Step-by-Step Procedure**

To provision the service:

1. If it is not already configured, configure NETCONF service over SSH on any devices where you are provisioning the new service.

   ```
   [edit]
   user@host2# set system services netconf ssh
   user@host2# commit
   ```

2. From a configuration management server, establish a NETCONF session with the device where you are provisioning the service.

   ```
   %ssh -p 830 -s user@host2 netconf
   user@host2's password:

   <!-- user user, class super-user -->
   <hello>
     <capabilities>
      <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
      </capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
      </capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:validate:1.0
      </capability>
      <capability>
        urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file

       </capability>
      <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
      <capability>http://xml.juniper.net/dmi/system/1.0</capability>
     </capabilities>
     <session-id>28898</session-id>
   </hello>
   ]]>]]>
   ```

3. If you are provisioning a new service on the device, enter a remote procedure call (RPC) that calls the service op script using the **create** action, and include values for all parameters that require configuring.

   The value for the **service-id** parameter should be identical to the service name.

   ```
   <rpc>
    <op-script>
     <script>vpn-service</script>
     <action>create</action>
     <service-id>vpn-service</service-id>
     <ifname>ge-2/0/5,ge-2/0/6</ifname>
     <ifdesc>connected to customer1-site-1,connected to customer3-site-2</ifdesc>

    </op-script>
   </rpc>
   ```

*Updating or Deleting Services Using NETCONF*

**Step-by-Step Procedure**

To update or delete an existing service:

1. If it is not already configured, configure NETCONF service over SSH on any devices where you are updating or deleting the service.

   ```
   [edit]
   user@host2# set system services netconf ssh
   user@host2# commit
   ```

2. From a configuration management server, establish a NETCONF session with the device where you are provisioning the service.

   ```
   %ssh -p 830 -s user@host2 netconf
   user@host2's password:

   <!-- user user, class super-user -->
   <hello>
     <capabilities>
      <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
      </capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0
      </capability>
      <capability>
         urn:ietf:params:xml:ns:netconf:capability:validate:1.0
      </capability>
      <capability>
        urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file

       </capability>
      <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
      <capability>http://xml.juniper.net/dmi/system/1.0</capability>
     </capabilities>
     <session-id>28898</session-id>
   </hello>
   ]]>]]>
   ```

3. If the given service is already provisioned on the device and you are updating the service, enter an RPC that calls the service op script using the **update** action, and include values for all parameters that require updating.

   ```
   <rpc>
    <op-script>
     <script>vpn-service</script>
     <action>update</action>
     <service-id>vpn-service</service-id>
     <ifname>ge-2/0/5</ifname>
     <ifdesc>connected to customer1-site-2</ifdesc>
    </op-script>
   </rpc>
   ```

4. If the given service is already provisioned on the device and you are deleting some or all of the service parameters, enter an RPC that calls the service op script using the **delete** action, and include any parameters that need to be deleted.

```
<rpc>
 <op-script>
  <script>vpn-service</script>
  <action>update</action>
  <service-id>vpn-service</service-id>
  <ifname>ge-2/0/6</ifname>
 </op-script>
</rpc>
```

## Verification

Confirm that the configuration is updated.

- Verifying the Service Configuration on page 256
- Verifying the Service Configuration on page 256

***Verifying the Service Configuration***

**Purpose** Verify that the commit is successful.

**Action** Issue the **show system commit** operational mode command to view the recent commits. The most recent commit entry shows that a commit was made through the NETCONF server by **user**.

```
user@host2> show system commit

0   2012-05-21 12:15:08 PDT by user via junoscript
1   2012-05-18 09:47:40 PDT by user via other
...
```

***Verifying the Service Configuration***

**Purpose** Verify that the service configuration is present in the active configuration.

**Action** Issue the **show configuration | compare rollback** *num* operational mode command to view configuration changes.

```
user@host2> show configuration | compare rollback 1

[edit interfaces]
+   ge-2/0/5 {
+       description "connected to customer1-site-1";
+   }
+   ge-2/0/6 {
+       description "connected to customer3-site-2";
+   }
```

**Meaning** A comparison of the current configuration with the previous configuration shows that the interfaces and interface descriptions were added to the configuration.

## Troubleshooting



### *Troubleshooting a Failed Commit*

**Problem** You see the following message when creating, updating, or deleting a service on a device through a NETCONF session:

```
<output>
configuration database modified
</output>
```

The configuration has previously uncommitted changes, and the service script cannot commit the service configuration changes.

**Solution** Commit the previous changes or roll back the configuration as appropriate, and then resubmit the service configuration changes.

### *Troubleshooting a Failed Attempt to Delete Service Parameters*

**Problem** You see the following message when deleting a service parameter on a device through NETCONF:

```
<xnm:error xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
  <source-daemon>
    op-script
  </source-daemon>
  <message>
    xsl:attribute: Cannot add attributes to an element if children have been
already added to the element.
  </message>
</xnm:error>
```

**Solution** The RPC might include both the parameter and a child element. Remove the child element from the RPC.

PART 3

# Configuration Automation

CHAPTER 14

# Commit Scripts Overview

This chapter includes the following topics:



## 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" on page 19. For more information about SLAX, see "SLAX Overview" on page 35

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 on page 15
- SLAX Overview on page 35
- XSLT Overview on page 19

## 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 307.

**Related Documentation**

- Commit Script Overview on page 261

## 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

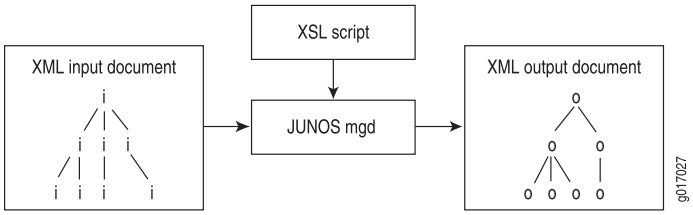
enforce custom configuration rules. Commit scripts are invoked during the commit process before the standard Junos OS validity checks are performed.

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 4 on page 264 shows the flow of commit script input and output.

### Figure 4: 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 264
- Commit Script Output on page 265
- Commit Scripts and the Junos OS Commit Model on page 266

## 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 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 on page 127

## 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.

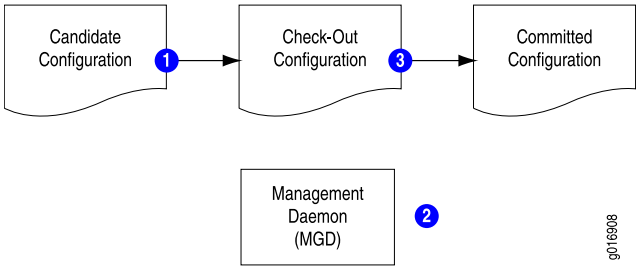
When you commit a set of changes in the candidate configuration, two methods are used to forward these changes to the current configuration:

- 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 5 on page 267 and the accompanying discussion explain how the standard commit model works:

Figure 5: Standard Commit Model



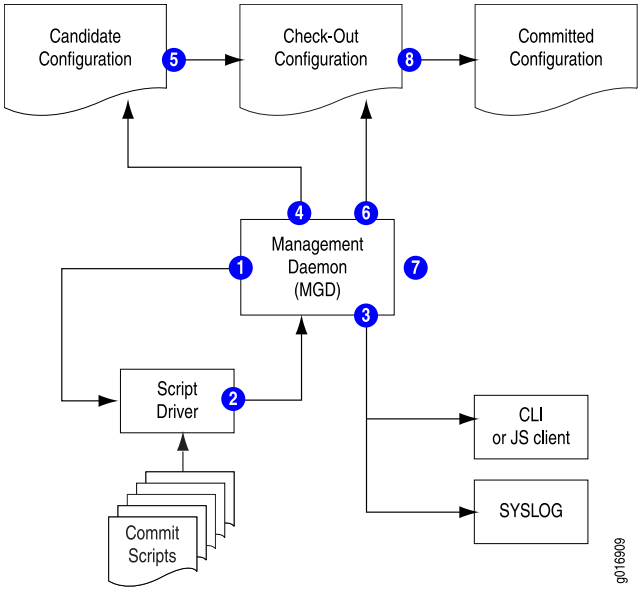

In the standard commit model, the software performs the following steps:

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 6 on page 267 and the accompanying discussion explain this process.

Figure 6: 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 269.

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 279), 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 307.

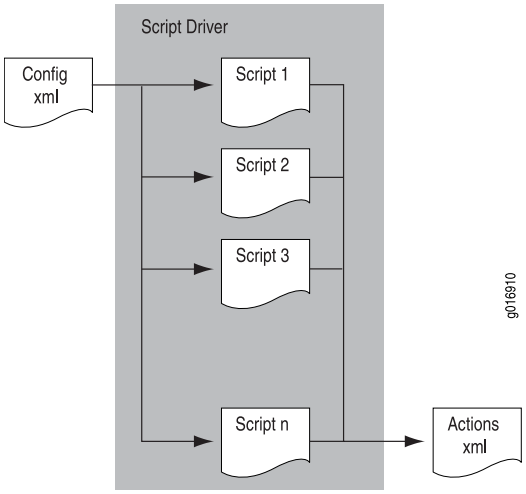
**Related Documentation**

- Avoiding Potential Conflicts When Using Multiple Commit Scripts on page 269

## 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 7 on page 269.

### Figure 7: 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 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 263

## 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: Understanding Extension Functions in the jcs and slax Namespaces" on page 95 and "Junos Script Automation: Named Templates in the jcs Namespace Overview" on page 124.

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 the 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" on page 124 and "Junos Named Templates in the jcs Namespace Summary" on page 125.

```
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" on page 22.

```
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 26 on page 272 summarizes the tasks and the syntax for each task.

### Table 26: 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> |
| Unprotect a statement or node in the hierarchy | unprotect="unprotect" | <address unprotect="unprotect"><br><name>192.168.1.1/30</name><br></address> |

Table 26: XML Syntax for Common Commit Script Tasks *(continued)*

| Action | Syntax | Example |
|---|---|---|
| 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 401.)

```
[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 on page 275
- Applying a Change to ISO-Enabled Interfaces on page 276

## Applying a Change to SONET/SDH Interfaces

The following commit script applies a transient change to each interface whose name begins with **[edit protocols mpls interface]**, setting the encapsulation to **ppp**. For information about transient changes, see "Overview of Generating Persistent or Transient Configuration Changes" on page 307. For a SLAX version of this example, see "Example: Generating a Transient Change" on page 322.

```
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 270 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 379.

```
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 270 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 322

CHAPTER 15

# Configuring and Troubleshooting Commit Scripts

At commit time, the Junos OS management process (mgd) looks in the **/config/scripts/commit** or the **/var/db/scripts/commit** directory, depending on whether the scripts are stored on the flash drive or the hard drive, for one or more commit scripts. Each commit script executes against the candidate configuration database to ensure the configuration conforms to the rules dictated by the scripts.

This chapter discusses the following topics:



## 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" on page 213 and "Storing Scripts in Flash Memory" on page 214. 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 280
- Preventing Commit Scripts from Executing During Commit Operations on page 281
- Deactivating Commit Scripts on page 281
- Activating Commit Scripts on page 282


## 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" on page 214.
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 on page 677
- Configuring Checksum Hashes for an Op Script on page 475
- 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;
```

## Displaying Commit Script Output

Table 27 on page 283 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 285.

NOTE: Tracing commit script processing, including the **cscript.log** file, is not supported on the QFX3000-G QFabric system.

**Table 27: Commit Script Configuration and Operational Mode Commands**

| Task | Command |
|---|---|
| Configuration Mode Commands | |
| Display errors and warnings generated by commit scripts. | **commit** or **commit check** |

Table 27: Commit Script Configuration and Operational Mode Commands *(continued)*

| Task | Command |
|---|---|
| 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** |
| 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 285

## 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 on page 285
- Configuring Tracing of Commit Scripts on page 287

### 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 28 on page 286 summarizes useful filtering commands that display selected portions of the **cscript.log** file.

### Table 28: 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 on page 287
- Configuring the Number and Size of Commit Script Log Files on page 287
- Configuring Access to Commit Script Log Files on page 288
- Configuring the Commit Script Trace Operations on page 288

### 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 29 on page 288 describes the meaning of the commit script tracing flags.

**Table 29: 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 |

Table 29: Commit Script Tracing Flags *(continued)*

| Flag | Description | Default Setting |
|---|---|---|
| **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 30 on page 289 describes some common problems that might occur.

Table 30: 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. |

Table 30: Troubleshooting Commit Scripts *(continued)*

| Problem | Solution |
|---|---|
| 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. |
| | 2. Make sure you have included the required boilerplate in your script. For more information, see "Required Boilerplate for Commit Scripts" on page 270. |
| | 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. |
| | 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>`<xsl:for-each select="interfaces/interface[starts-with(name, 't1-')]">`<br>`<xsl:if test="not(description)">` |

CHAPTER 16

# Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message

This chapter discusses the following topics:



## 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 307.

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 301
- Example: Generating a Custom System Log Message on page 304
- Example: Generating a Custom Warning Message on page 298
- Generating a Custom Warning, Error, or System Log Message on page 292
- Tag Elements to Use When Generating Messages on page 295

## 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 270. 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" on page 30. For

information about writing scripts in SLAX instead of XSLT, see "SLAX Overview" on page 35.

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 307.)

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 295.

For SLAX versions of these constructs, see "Example: Generating a Custom Warning Message" on page 298, "Example: Generating a Custom Error Message" on page 301, and "Example: Generating a Custom System Log Message" on page 304.

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" on page 214.

   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 301
- Example: Generating a Custom System Log Message on page 304
- Example: Generating a Custom Warning Message on page 298

## Tag Elements to Use When Generating Messages

Table 31 on page 295 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 298, "Example: Generating a Custom Error Message" on page 301, and "Example: Generating a Custom System Log Message" on page 304.

### Table 31: 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. |

Table 31: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages *(continued)*

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|---|---|---|
| **<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. |
| **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. |

Table 31: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages *(continued)*

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|---|---|---|
| **<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. |
| **<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**. |

## Examples: Generating Custom Warning, Error, and System Log Messages


- Example: Generating a Custom Warning Message on page 298
- Example: Generating a Custom Error Message on page 301
- Example: Generating a Custom System Log Message on page 304

## 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 292.)

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 301
- Example: Generating a Custom System Log Message on page 304
- Generating a Custom Warning, Error, or System Log Message on page 292

## 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 304
- Example: Generating a Custom Warning Message on page 298
- Generating a Custom Warning, Error, or System Log Message on page 292

## 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

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 **show log *filename*** operational mode command. For example, if messages are logged to the **messages** file, issue the following command:

```
user@host> show 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 301
- Example: Generating a Custom Warning Message on page 298
- Generating a Custom Warning, Error, or System Log Message on page 292

CHAPTER 17

# Writing Commit Scripts That Generate a Persistent or Transient Configuration Change

This chapter discusses the following topics:



## 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 32 on page 309.

### Table 32: 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. |
| 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. |

Table 32: Differences Between Persistent and Transient Changes *(continued)*

| Persistent Changes | Transient Changes |
|---|---|
| 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. |
| 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 266. |
| 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 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*.

## 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 270. 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" on page 30. For information about writing scripts in SLAX instead of XSLT, see "SLAX Overview" on page 35.

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 291.

```
<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" on page 214.

   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" on page 214.) 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 33 on page 317 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" on page 318.

### Table 33: Tags and Attributes for Creating Configuration Changes

| Data Item, XML Element, or Attribute | Description |
|---|---|
| Container Tags | |

Table 33: Tags and Attributes for Creating Configuration Changes *(continued)*

| Data Item, XML Element, or Attribute | Description |
|---|---|
| **<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. |

## Examples: Generating Persistent and Transient Changes



### 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

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 ...
```

## 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**

- Generating a Persistent or Transient Change on page 311
- Overview of Generating Persistent or Transient Configuration Changes on page 307
- Removing a Persistent or Transient Change on page 316
- **jcs:emit-change Template on page 127**

CHAPTER 18

# Writing Commit Scripts That Create Custom Configuration Syntax with Macros

This chapter discusses the following topics:



## 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 307). 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 355.

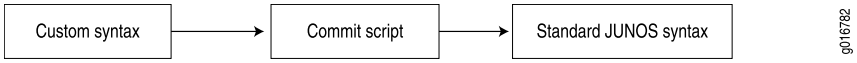
**Related Documentation**


- How Macros Work on page 328
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 333
- Example: Creating Custom Configuration Syntax with Macros on page 335

## 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 8 on page 328. The standard Junos OS statements are added to the configuration to cause your intended operational changes.

### Figure 8: 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 335.

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 328 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 328, 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 383.

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 327
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 333
- Example: Creating Custom Configuration Syntax with Macros on page 335

## 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 270. 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 335.

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" on page 214.

   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 327
- How Macros Work on page 328
- Example: Creating Custom Configuration Syntax with Macros on page 335

## Example: Creating Custom Configuration Syntax with Macros

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



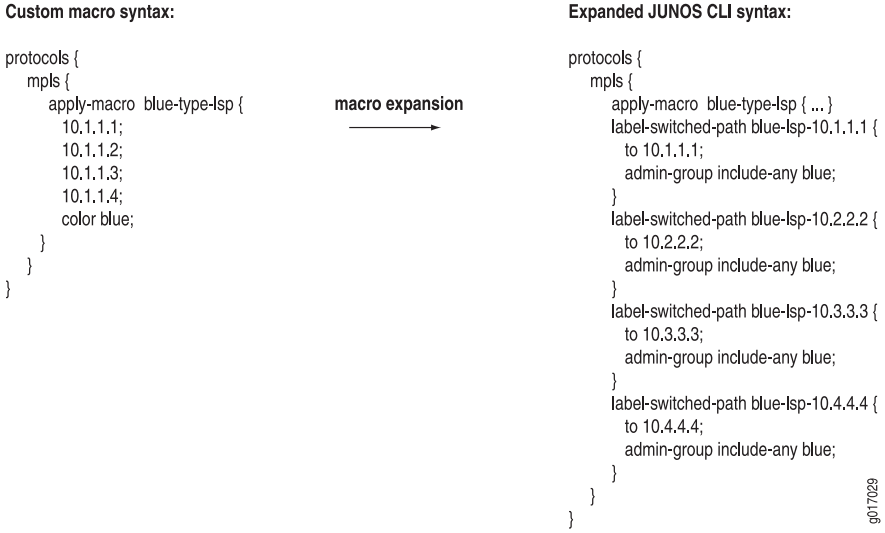
### Requirements

This example uses a device running Junos OS.

### Overview and Commit Script

Figure 9 on page 335 shows a macro that uses custom syntax and the corresponding expansion to standard Junos OS command-line interface (CLI) syntax.

**Figure 9: Sample Macro and Corresponding Junos OS CLI Expansion**




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 9 on page 335 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 9 on page 335 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 9 on page 335 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 362.

## 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 327
- How Macros Work on page 328
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 333

CHAPTER 19

# Commit Script Examples

This chapter includes examples for commit scripts. Each example provides a brief overview, the script, configuration instructions, and verification of the script and configuration. The scripts are provided in both Extensible Stylesheet Language Transformations (XSLT) and Stylesheet Language Alternative Syntax (SLAX).

## 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 352
- Overview and Commit Script on page 352

## 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 335.

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 on page 398
- Overview and Commit Script on page 398

## 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 20

# Summary of Junos XML and XSLT Tag Elements Used in Commit Scripts

This chapter lists the Junos XML tag elements used to generate custom warning, error, and system log (syslog) messages, and the XSLT tag elements used to make custom permanent or transient changes. The tag elements are in alphabetical order.

## <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 307 and "Overview of Creating Custom Configuration Syntax with Macros" on page 327.

**Related Documentation**

- <transient-change> (XSLT) on page 440

## <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 292.

## <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 307 and "Overview of Creating Custom Configuration Syntax with Macros" on page 327.

**Related Documentation**

- <change> (XSLT) on page 439

## 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 292.

**Related Documentation**

- xnm:warning (Junos XML) on page 443

## 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 292

**Related Documentation**

- xnm:error (Junos XML) on page 442.

CHAPTER 21

# Summary of Commit Script Configuration Statements

This chapter describes each configuration statement for commit scripts. The statements are organized alphabetically.

## allow-transients

**Syntax** allow-transients;

**Hierarchy Level** [edit system scripts commit]

**Release Information** Statement introduced in Junos OS Release 7.4.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 311
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 333

## 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.
Statement introduced in Junos OS Release 12.2 for the QFX Series.

**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 327

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 282
- Configuring Checksum Hashes for an Event Script on page 677
- Configuring Checksum Hashes for an Op Script on page 475
- Executing an Op Script from a Remote Site on page 476
- 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;
  }
  max-datasize
  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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 on page 213

## direct-access

**Syntax** direct-access;

**Hierarchy Level** [edit system scripts commit]

**Release Information** Statement introduced in Junos OS Release 9.1.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 283

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 279

## max-datasize

**Syntax** max-datasize *size*;

**Hierarchy Level** [edit event-options event-script],
[edit system scripts commit],
[edit system scripts op]

**Release Information** Statement introduced in Junos OS Release 12.3.

**Description** Maximum amount of memory allocated for the data segment during execution of a script of the given type. Junos OS sets the maximum memory limit for the executing script to the configured value irrespective of the total memory available on the system at the time of execution. If the executing script exceeds the specified maximum memory limit for that script type, it exits gracefully.

**Default** If you do not include the **max-datasize** statement, the system allocates half of the total available memory of the system up to a maximum value of 128 MB for the data segment portion of the executed script.

**Options** ***size***—Maximum amount of memory allocated for the data segment during execution of a script of the given type. If you do not specify a unit of measure, the default is bytes.

**Syntax:** ***size*** to specify bytes, ***size*k** to specify KB, ***size*m** to specify MB, or ***size*g** to specify GB

**Range:** 2,3068,672 (22 MB) through 1,073,741,824 (1 GB)

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- max-policies on page 640
- Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts on page 227

## optional

**Syntax** optional;

**Hierarchy Level** [edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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.

NOTE: On the QFabric system, commit scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/commit/** directory on the Director 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 279

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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.

NOTE: On the QFabric system, commit scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/commit/** directory on the Director device.

**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 on page 218
- refresh-from (Commit Scripts) on page 453
- source (Commit Scripts) on page 455

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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.

NOTE: This statement is not supported on the QFabric system.

**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 on page 222
- refresh (Commit Scripts) on page 452
- source (Commit Scripts) on page 455

## scripts

**Syntax**

```
scripts {
  commit {
    allow-transients;
    direct-access;
    file filename {
      checksum (md5 | sha-256 | sha1) hash;
      optional;
      refresh;
      refresh-from url;
      source url;
    }
    max-datasize
    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;
      max-datasize
      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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS commit or op scripts, configure scripting mechanisms.

NOTE: The **traceoptions** statement is not supported on QFabric systems.

**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 on page 213

## source (Commit Scripts)

**Syntax** source *url*;

**Hierarchy Level** [edit system scripts commit file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.4.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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.

NOTE: On the QFabric system, commit scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/op/** directory on the Director device.

**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 on page 218
- Overview of Updating Scripts from a Remote Source on page 216
- **refresh (Commit Scripts) on page 452**
- **refresh-from (Commit Scripts) on page 453**

## 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**

**file *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.

**Default:** Commit scripts: /var/log/cscript.log, Op scripts: /var/log/op-script.log

**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 *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:** *size* to specify bytes, *size*k to specify KB, *size*m to specify MB, or *size*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 285
- Tracing Op Script Processing on page 478

PART 4

# Operations Automation

CHAPTER 22

# Operation (Op) Scripts Overview

This chapter includes the following topics:



## Op Script Overview

Junos OS operation (op) scripts automate network and device management and troubleshooting. Op scripts can perform any function available through the remote procedure calls (RPCs) supported by either the Junos XML management protocol or the Junos Extensible Markup Language (XML) API. Op scripts can be executed manually in the CLI or upon user login, or they can be called from another script. They are executed by the Junos OS management (mgd) process.

Op scripts enable you to do the following things:

- Create custom operational mode commands
- Execute a series of operational mode commands
- Customize the output of operational mode commands
- Shorten troubleshooting time by gathering operational information and iteratively narrowing down the cause of a network problem
- Perform controlled configuration changes
- Monitor the overall status of a device by creating a general operation script that periodically checks network warning parameters, such as high CPU usage.

Op scripts are based on the Junos XML management protocol, and the Junos XML API, which are discussed in "Junos XML API and Junos XML Management Protocol Overview" on page 15. Op scripts can be written in either the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) scripting language. Op scripts use XPath to locate the operational objects to be inspected and XSLT constructs to specify the actions to perform on the located operational objects. The actions can change the output or execute additional commands based on the output.

**Related Documentation**

- SLAX Overview on page 35
- XPath Overview on page 22

- XSLT Overview on page 19

## How Op Scripts Work

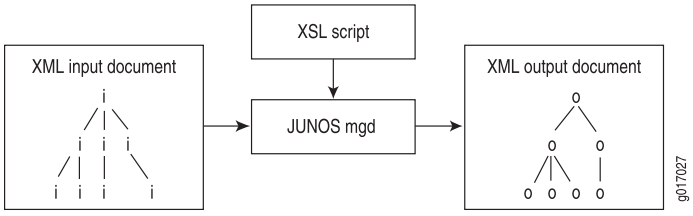
Op scripts execute Junos OS operational commands and inspect the resulting output. After inspection, op scripts can automatically correct errors within the device running Junos OS based on this output.

You add op scripts to device operations by listing the filenames of one or more op script files within the **[edit system scripts op]** hierarchy level. These files must be added to the appropriate op script file directory. For more information about op script file directories, see "Storing Scripts in Flash Memory" on page 214. Once added to the device, op scripts are invoked from the command line, using the **op *filename*** command.

You can use op scripts to generate changes to the device configuration by including the **<load-configuration>** tag element. 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 10 on page 462 shows a high-level view of the flow of op script input and output.

### Figure 10: Op Script Input and Output



CHAPTER 23

# Writing Op Scripts

This chapter explains how to write operation (op) scripts and includes the following topics:



## Required Boilerplate for Op Scripts

When you write operation (op) 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 op scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make 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: Understanding Extension Functions in the jcs and slax Namespaces" on page 95 and "Junos Script Automation: Named Templates in the jcs Namespace Overview" on page 124.

Op 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 op scripts that you create. The XSLT boilerplate follows:

**XSLT Boilerplate for Op 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="/">
9       <op-script-results>
        <!-- ... insert your code here ... -->
10      </op-script-results>
11    </xsl:template>
    <!-- ... insert additional template definitions here ... -->
12  </xsl:stylesheet>
```

Line 1 is the Extensible Markup Language (XML) processing instruction (PI), which marks this file as XML and specifies the version of XML as 1.0. The XML PI, if present, must be the first non-comment token in the script file.

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

Line 2 opens the style sheet and specifies the XSLT version as 1.0.

```
2  <xsl:stylesheet version="1.0"
```

Lines 3 through 6 list all the namespace mappings commonly used in operation 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.

```
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 (in the file **/usr/libdata/cscript/import/junos.xsl**). 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" on page 124 and "Junos Named Templates in the jcs Namespace Summary" on page 125.

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

Line 8 defines a template that matches the **</>** element. The **<xsl:template match="/">** element is the root element and represents the top level of the XML hierarchy. All XML Path Language (XPath) expressions in the script must start at the top level. This allows the script to access all possible Junos XML and Junos XML protocol remote procedure calls (RPCs). For more information, see "XPath Overview" on page 22.

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

After the **<xsl:template match="/">** tag element, the **<op-script-results>** and **</op-script-results>** container tags must be the top-level child tags, as shown in Lines 9 and 10.

```
9       <op-script-results>
        <!-- ... insert your code here ... -->
10      </op-script-results>
```

Line 11 closes the template.

```
11    </xsl:template>
```

Between Line 11 and Line 12, you can define additional XSLT templates that are called from within the **<xsl:template match="/">** template.

Line 12 closes the style sheet and the op script.

```
12  </xsl:stylesheet>
```

**SLAX Boilerplate for Op 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 / {
  <op-script-results> {
    /*
     * Insert your code here
    */
  }
}
```

## Mapping Operational Mode Commands and Output Fields to Junos XML Notation

In op scripts, you use tag elements from the Junos XML API to represent operational mode commands and output fields. For the Junos XML equivalent of commands and output fields, consult the *Junos XML API Operational Reference*.

You can also display the Junos XML tag elements for operational mode command output by directing the output from the command to the **| display xml** command:

```
user@host> command-string | display xml
```

For example:

```
user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
    <interface-information
xmlns="http://xml.juniper.net/junos/10.0RI0/junos-interface" junos:style="terse">
        <physical-interface>
            <name>dsc</name>
            <admin-status>up</admin-status>
            <oper-status>up</oper-status>
        </physical-interface>
        <physical-interface>
            <name>fxp0</name>
            <admin-status>up</admin-status>
            <oper-status>up</oper-status>
            <logical-interface>
                <name>fxp0.0</name>
                <admin-status>up</admin-status>
                <oper-status>up</oper-status>
                ...
```

**Related Documentation**

- Configuring Help Text for Op Scripts on page 471
- Declaring Arguments in Op Scripts on page 469
- Using RPCs and Operational Mode Commands in Op Scripts on page 466

## Using RPCs and Operational Mode Commands in Op Scripts

Most Junos OS operational mode commands have XML equivalents. These XML commands can be executed remotely using the remote procedure call (RPC) protocol. All operational mode commands that have XML equivalents are listed in the *Junos XML API Operational Reference*.

Use of RPC and operational mode commands in op scripts is discussed in more detail in the following sections:

- Using RPCs in Op Scripts on page 466
- Displaying the RPC Tags for a Command on page 467
- Using Operational Mode Commands in Op Scripts on page 468

### Using RPCs in Op Scripts

To use an RPC in an op script, include the RPC in a variable declaration. You then invoke the RPC with the **jcs:invoke()** or **jcs:execute()** extension function and include the RPC variable as an argument. The **jcs:invoke()** function executes the RPC on the local device. You can use the **jcs:execute()** function with a connection handle to execute the RPC on a remote device.

The following snippet, which invokes an RPC on the local device, is expanded and fully described in "Example: Customizing Output of the show interfaces terse Command Using an Op Script" on page 497.

**XSLT Syntax**

```
<xsl:variable name="rpc">
  <get-interface-information/> # Junos RPC for the show interfaces command
</xsl:variable>
<xsl:variable name="out" select="jcs:invoke($rpc)"/>
...
```

**SLAX Syntax**

```
var $rpc = <get-interface-information>;
var $out = jcs:invoke($rpc);
```

To execute an RPC on a remote device, an SSH session must be established. In order for the script to establish the connection, you must either configure the SSH host key information for the remote device on the local device where the script will be executed, or the SSH host key information for the remote device must exist in the known hosts file of the user executing the script. For each remote device where an RPC is executed, configure the SSH host key information with one of the following methods:

- To configure SSH known hosts on the local device, include the **host** statement, and specify hostname and host key options for the remote device at the **[edit security ssh-known-hosts]** hierarchy level of the configuration.
- To manually retrieve SSH host key information, issue the **set security ssh-known-hosts fetch-from-server *hostname*** configuration mode command to instruct Junos OS to connect to the remote device and add the key.

  ```
  user@host# set security ssh-known-hosts fetch-from-server router2
  The authenticity of host 'router2 (10.10.10.1)' can't be established.
  RSA key fingerprint is 30:18:99:7a:3c:ed:40:04:0f:fd:c1:57:7e:6b:f3:90.
  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added 'router2,10.10.10.1' (RSA) to the list of known
  hosts.
  ```

- To manually import SSH host key information from a file, use the **set security ssh-known-hosts load-key-file *filename*** configuration mode command and specify the known-hosts file.

  ```
  user@host# set security ssh-known-hosts load-key-file /var/tmp/known_hosts
  Import SSH host keys from trusted source /var/tmp/known_hosts ? [yes,no] (no)
  yes
  ```

- Alternatively, the user executing the script can log in to the local device, SSH to the remote device, and then manually accept the host key, which is added to that user's known hosts file. In the following example, root is logged in to router1. In order to execute a remote RPC on router2, root adds the host key of router2 by issuing the **ssh router2** operational mode command and manually accepting the key.

  ```
  root@router1> ssh router2
  The authenticity of host 'router2 (10.10.10.1)' can't be established.
  RSA key fingerprint is 30:18:99:7a:3c:ed:40:04:0f:fd:c1:57:7e:6b:f3:90.
  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added 'router2,10.10.10.1' (RSA) to the list of known
  hosts.
  ```

## Displaying the RPC Tags for a Command

To display the remote procedure call (RPC) XML tags for an operational mode command, enter **display xml rpc** after the pipe symbol ( | ).

The following example displays the RPC tags for the **show route** command:

```
user@host> show route | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.1I0/junos">
    <rpc>
        <get-route-information>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

## Using Operational Mode Commands in Op Scripts

Some operational mode commands do not have XML equivalents. If a command is not listed in the *Junos XML API Operational Reference*, the command does not have an XML equivalent.

Another way to determine whether a command has an XML equivalent is to issue the command followed by the **| display xml** command:

user@host> ***operational-mode-command*** **| display xml**

If the output includes only tag elements like **<output>**, **<cli>**, and **<banner>**, the command might not have an XML equivalent. In the following example, the output indicates that the **show host** command has no XML equivalent:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
    <output>
       ...
    </output>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

NOTE: For some commands that have an XML equivalent, the output of the piped **| display xml** command does not include tag elements other than **<output>**, **<cli>**, and **<banner>** only because the relevant feature is not configured. For example, the **show services cos statistics forwarding-class** command has an XML equivalent that returns output in the **<service-cos-forwarding-class-statistics>** response tag, but if the configuration does not include include any statements at the **[edit class-of-service]** hierarchy level then there is no actual data for the **show services cos statistics forwarding-class | display xml** command to display. The output is something like this:

```
user@host> show services cos statistics forwarding-class | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/8.3IO/junos">

    <cli>

        <banner></banner>

    </cli>

</rpc-reply>
```

For this reason, the information in the *Junos XML API Operational Reference* is normally more reliable.

An op script can include commands that have no XML equivalent. Use the **<command>**, **<xsl:value-of>**, and **<output>** elements in the script, as shown in the following code snippet. This snippet is expanded and fully described in "Example: Displaying DNS Hostname Information Using an Op Script" on page 507.

```
<xsl:variable name="query">
  <command>
    <xsl:value-of select="concat('show host ', $hostname)"/>
  </command>
</xsl:variable>
<xsl:variable name="result" select="jcs:invoke($query)"/>
<xsl:variable name="host" select="$result"/>
<output>
  <xsl:value-of select="concat('Name: ', $host)"/>
</output>
...
```

**Related Documentation**

- Configuring Help Text for Op Scripts on page 471
- Declaring Arguments in Op Scripts on page 469
- Mapping Operational Mode Commands and Output Fields to Junos XML Notation on page 465

## Declaring Arguments in Op Scripts

There are two ways to declare arguments to an op script: by including XSLT or SLAX instructions in the script or by including statements in the Junos configuration. *Script-generated* and *configuration-generated* arguments have the same operational impact.

To declare arguments within a script, declare a global variable named **arguments**, containing **<argument>** tag elements. Within each **<argument>** tag element, include the required **<name>** tag element and the optional **<description>** tag element:

**XSLT Syntax**

```
<xsl:variable name="arguments">
  <argument>
    <name>name</name>
    <description>name</description>
  </argument>
</xsl:variable>
```

**SLAX Syntax**

```
var $arguments = {
  <argument> {
    <name> "name";
    <description> "descriptive-text";
  }
}
```

To declare arguments in the configuration, include the **arguments** statement in the **[edit system scripts op file *filename*]** hierarchy level.

```
[edit system scripts op file filename]
arguments {
```

```
    argument-name {
      description descriptive-text;
    }
  }
```

If you include the optional **<description>** tag element or the **description** statement, the text of the description appears in the command-line interface (CLI) as a help-text string to describe the purpose of the argument, as discussed in "Configuring Help Text for Op Scripts" on page 471.

In the operation script, you must include a corresponding parameter declaration for each argument. The parameter name must match the name of the argument:

```
<xsl:param name="name"/>
```

The SLAX equivalent is:

```
param $name;
```

You can create a hidden argument by including the **<xsl:param name="*name*"/>** instruction without listing the argument in the **arguments** variable or in the configuration.

After you declare an argument, you can use command completion to list available arguments:

```
user@host> op filename ?
Possible completions:
  argument-name        description
  argument-name        description
```

For each argument you include on the command-line, you must specify a corresponding value. To do this, include an ***argument-name*** and an ***argument-value*** when you execute the script with the **op *filename*** command:

```
user@host> op filename argument-name argument-value
```

NOTE: If you specify an argument that the script does not recognize, the script ignores the argument.

If you configure arguments by including the **arguments** statement in the configuration, any arguments that you declare directly in the script are still available, but are not listed among the **Possible completions** when you issue the **op *filename* ?** command.

If you declare all arguments in the script (and none in the configuration), then the arguments do appear in the **Possible completions** list. This is because the management (mgd) process populates the **Possible completions** list by first checking the configuration for arguments. The mgd process looks in the script for arguments only if no arguments are found in the configuration. Thus, if arguments are declared in the configuration, the arguments declared in the script become hidden in the CLI.

### Example: Declaring Arguments

Declare two arguments named **interface** and **protocol**. Execute the script, specifying the ge-0/2/0.0 interface and the **inet** protocol as values for the arguments. For either method, you must declare corresponding script parameters:

```
<xsl:param name="interface"/>
<xsl:param name="protocol"/>
```

**Declaring Arguments in the Op Script (script1)**

```
<xsl:variable name="arguments">
  <argument>
    <name>interface</name>
    <description>Name of interface to display</description>
  </argument>
  <argument>
    <name>protocol</name>
    <description>Protocol to display (inet, inet6)</description>
  </argument>
</xsl:variable>
```

**Declaring Arguments in the Configuration**

```
[edit system scripts op]
file script1 {
  arguments {
    interface {
      description "Name of interface to display";
    }
    protocol {
      description "Protocol to display (inet, inet6)";
    }
  }
}
```

**Executing the Script**

```
user@host> op script1 interface ge-0/2/0.0 protocol inet
```

**Related Documentation**

- Example: Importing Files Using an Op Script on page 523
- Configuring Help Text for Op Scripts on page 471
- Mapping Operational Mode Commands and Output Fields to Junos XML Notation on page 465
- Using RPCs and Operational Mode Commands in Op Scripts on page 466

## Configuring Help Text for Op Scripts

You can provide help text to describe an op script and its arguments when the **?** is used to list possible completions in the CLI. Include the **description** statement:

```
description descriptive-text;
```

You can include this statement at the following hierarchy levels:

- **[edit system** scripts op file *filename*]

- **[edit system** scripts op file *filename* arguments *argument-name*]

The following examples show the configuration and the resulting output.

## Examples: Configuring Help Text for Op Scripts

Configure help text for a script and display the resulting output:

```
[edit system scripts op]
user@host# set file interface.xsl description "Test the interface"
user@host# commit
...
[edit system scripts op]
user@host# set file ?
Possible completions:
 <name>          Local filename of the script file
 interface.xsl    Test the interface
```

Configure help text for a script's arguments and display the resulting output:

```
[edit system scripts op file interface.xsl arguments]
user@host# set t1 description "Search for T1 interfaces"
user@host# set t3 description "Search for T3 interfaces"
user@host# commit
...
[edit system scripts op file interface.xsl arguments]
user@host# set ?
Possible completions:
 <name>       Name of the argument
 t1           Search for T1 interfaces
 t3           Search for T3 interfaces
```

**Related Documentation**

- Declaring Arguments in Op Scripts on page 469
- Mapping Operational Mode Commands and Output Fields to Junos XML Notation on page 465
- Using RPCs and Operational Mode Commands in Op Scripts on page 466

CHAPTER 24

# Configuring and Executing Op Scripts

Operation (op) scripts allow you to automate network troubleshooting and network management. This chapter discusses command-line interface (CLI) configuration statements and operational mode commands for enabling and executing op scripts.

To configure op scripts, include the following statements at the **[edit]** hierarchy level:

```
system {
  scripts {
    op {
      file filename {
        arguments {
          argument-name {
            description descriptive-text;
          }
        }
        checksum (md5 | sha-256 | sha1) hash;
        command filename-alias;
        description descriptive-text;
        max-datasize
        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;
      }
    }
  }
}
```

This chapter discusses the following topics:

- Tracing Op Script Processing on page 478
- Disabling an Op Script on page 481

## Enabling an Op Script and Defining a Script Alias

Operation (op) scripts are stored on a device's hard drive in the **/var/db/scripts/op** directory or on the flash drive in the **/config/scripts/op** directory. Only users in the Junos OS **super-user** login class can access and edit files in these directories. For information about setting the storage location for scripts, see "Storing Scripts in Flash Memory" on page 214.

NOTE: If the device has dual Routing Engines and you want to enable an op script to execute on both Routing Engines, you must copy the script to the **/var/db/scripts/op** or **/config/scripts/op** directory on both Routing Engines. The **commit synchronize** command does not automatically copy scripts between Routing Engines.

You must enable an op script before it can be executed. Include the **file *filename*** statement at the **[edit system scripts op]** hierarchy level, specifying the name of an XSLT or SLAX file containing an op script. Only users who belong to the Junos **super-user** login class can enable op scripts.

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

The filename of an op script written in SLAX must include the **.slax** extension for the script to be enabled and executed. No particular filename extension is required for op scripts written in XSLT, but we strongly recommend that you append the **.xsl** extension. Whether or not you choose to include the **.xsl** extension on the file, the filename that you add at the **[edit system scripts op]** hierarchy level must exactly match the filename of the script in the directory. For example, if the XSLT script filename is **script1.xsl**, then you must include **script1.xsl** in the configuration hierarchy to enable the script; likewise, if the XSLT script filename is **script1**, then you must include **script1** in the configuration hierarchy.

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

Optionally, you can define an alias for an op script and then specify either the filename or the alias when you execute the script. To define the alias, include the **command** statement at the **[edit system scripts op file *filename*]** hierarchy level:

```
[edit system scripts op]
file filename {
    command filename-alias;
}
```

## Configuring Checksum Hashes for an Op Script

You can configure one or more checksum hashes that can be used to verify the integrity of an op 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/op** 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/op/script1.slax) = 3af7884eb56e2d4489c2e49b26a39a97
   user@host> file checksum sha1 /var/db/scripts/commit/script1.slax
     SHA1 (/var/db/scripts/op/script1.slax) =
     00dc690fb08fb049577d012486c9a6dad34212c0
   user@host> file checksum sha-256 /var/db/scripts/commit/script1.slax
     SHA256 (/var/db/scripts/op/script1.slax) =
     150bf53383769f3bfedd41fe73320777f208d4fda81230cb27b8738
   ```

4. Configure the script with one or more hash values.

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

NOTE: If the op script is stored remotely, do not include the checksum statement in the configuration. You can verify the script's integrity before it runs by specifying the hash value on the command line when you run the op command with the <url> option and the <key> option.

**Related Documentation**

- Configuring Checksum Hashes for a Commit Script on page 282
- Configuring Checksum Hashes for an Event Script on page 677
- 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*
- op command in the *System Basics and Services Command Reference*

## Executing an Op Script

Unlike commit scripts, operation (op) scripts do not execute during a commit operation. When you issue the **commit** command, op scripts configured at the **[edit system scripts op]** hierarchy level are placed into system memory and enabled for execution. After the commit operation completes, you can execute an op script from the CLI by issuing the **op** operational mode command. You also can configure the device to execute an op script automatically when a member of a specified Junos OS login class logs in to the CLI.

### Executing an Op Script by Issuing the op Command

To execute an op script from the CLI, issue the **op** operational mode command, specifying a URL, the script filename, or the alias defined by the **command** statement at the **[edit system scripts op file *filename*]** hierarchy level.

```
user@host> op (filename-or-alias | url url)
```

### Executing an Op Script at Login

You can configure an op script to execute automatically when any user belonging to a designated Junos OS login class logs in to the CLI. To associate an op script with a login class, include the **login-script *script-filename*** statement at the **[edit system login class *class-name*]** hierarchy level:

```
[edit system login]
class class-name {
    login-script script-filename;
}
```

The following example configures the **super-user-login.slax** op script to execute when any user who belongs to the **super-user** class logs in to the CLI (provided that the script has been enabled as discussed in "Enabling an Op Script and Defining a Script Alias" on page 474).

```
[edit system login]
class super-user {
    login-script super-user-login.slax;
}
```

**Related Documentation**

- login-script (Login)

## Executing an Op Script from a Remote Site

As an alternative to storing operation (op) scripts locally on the device, you can store op scripts at a remote site. This allows you to execute the scripts by specifying a URL on the command line.

To execute an op script from a remote site:

1. Create the script.
2. (Optional) Store the script temporarily in the **/var/tmp** directory on the device, and 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/tmp/script1.slax
     MD5 (/var/tmp/script1.slax) = 3af7884eb56e2d4489c2e49b26a39a97
   user@host> file checksum sha1 /var/tmp/script1.slax
     SHA1 (/var/tmp/script1.slax) = 00dc690fb08fb049577d012486c9a6dad34212c0
   user@host> file checksum sha-256 /var/tmp/script1.slax
     SHA256 (/var/tmp/script1.slax) =
     150bf53383769f3bfedd41fe73320777f208d4fda81230cb27b8738
   ```

3. Place the script on the remote server.
4. Provide the script URL and the optional hash values to the administrators who will execute the script.
5. Execute the script by running the **op** command and specifying the URL that points to the remote file.

   ```
   user@host> op url https://www.juniper.net/scripts/2009-04-01.01.slax
     key md5 3af7884eb56e2d4489c2e49b26a39a97
   ```

   This example shows how to include the <key> option and the MD5 checksum information.

NOTE: If the op script is stored locally, do not include the hash key on the command line. Instead, configure the hash value by including the **checksum** statement at the **[edit system scripts op file *filename*]** hierarchy level. 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.

To prevent the execution of op scripts from remote sites, configure the **no-allow-url** statement at the **[edit system scripts op]** hierarchy level.

```
user@host# set system scripts op no-allow-url
```

When you configure the **no-allow-url** statement, issuing the **op url *url*** operational mode command generates an error.

**Related Documentation**

- Configuring Checksum Hashes for an Op Script on page 475
- 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*
- no-allow-url on page 542
- op command in the *System Basics and Services Command Reference*

## Tracing Op Script Processing

Op script tracing operations track all op 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 op script tracing is to log important events in a file called **op-script.log** located in the **/var/log** directory. When the file **op-script.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. The resulting files are **op-script.log.0.gz**, then **op-script.log.1.gz**, until there are 10 trace files. Then the oldest trace file (**op-script.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 Enabling Traceoptions for Op Scripts

If no op script trace options are configured, the simplest way to view the trace output of an op script is to configure the **output** trace flag and issue the **show log op-script.log | last** command. To do this, perform the following steps:

1. If you have not done so already, enable an op script by including the **file** statement at the **[edit system scripts op]** hierarchy level:

   ```
   [edit system scripts op]
   user@host# set file filename
   ```

2. Enable trace options by including the **traceoptions flag output** statement at the **[edit system scripts op]** hierarchy level:

   ```
   [edit system scripts op]
   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/op-script.log** file. At the end of the log is the output generated by the op script you enabled in Step 1. To display the end of the log, issue the **show log op-script.log | last** operational mode command:

   ```
   [edit]
   user@host# run show log op-script.log | last
   ```

Table 34 on page 479 summarizes useful filtering commands that display selected portions of the **op-script.log** file.

Table 34: Op Script Tracing Operational Mode Commands

| Task | Command |
|---|---|
| Display logging data associated with all op script processing. | **show log op-script.log** |
| Display processing for only the most recent operation. | **show log op-script.log \| last** |
| Display processing for script errors. | **show log op-script.log \| match error** |
| Display processing for a particular script. | **show log op-script.log \| match *filename*** |

### Example: Minimum Configuration for Enabling Traceoptions for Op Scripts

Display the trace output of the op script file **source-route.xsl**:

```
[edit]
system {
   scripts {
      op {
         file source-route.xsl;
         traceoptions flag output;
      }
   }
}
```

```
[edit]
user@host# commit
[edit]
user@host# run show log op-script.log | last
```

## Configuring Tracing of Op 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 op traceoptions]** hierarchy level:

```
[edit system scripts op 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 Op Script Log Filename on page 480
- Configuring the Number and Size of Op Script Log Files on page 480

### Configuring the Op Script Log Filename

By default, the name of the file that records trace output is **op-script.log**. You can specify a different name by including the **file** statement at the **[edit system scripts op traceoptions]** hierarchy level:

```
[edit system scripts op traceoptions]
file filename;
```

### Configuring the Number and Size of Op 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 op traceoptions file <*filename*>]** hierarchy level:

```
[edit system scripts op 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 Op Script Log Files

By default, access to the op 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 op traceoptions file <*filename*>]** hierarchy level.

```
[edit system scripts op traceoptions file <filename>]
(world-readable | no-world-readable);
```

The **no-world-readable** statement restricts op script log access to the owner. The **world-readable** statement enables unrestricted access to the op script log file.

### Configuring the Op 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 op traceoptions]** hierarchy level:

```
[edit system scripts op traceoptions]
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
```

Table 35 on page 481 describes the meaning of the op script tracing flags.

**Table 35: Op Script Tracing Flags**

| Flag | Description | Default Setting |
|---|---|---|
| **all** | Trace all operations. | Off |
| **events** | Trace important events. | On |
| **input** | Trace op script input data. | Off |
| **offline** | Generate data for offline development. | Off |
| **output** | Trace op script output data. | Off |
| **rpc** | Trace op script RPCs. | Off |
| **xslt** | Trace the Extensible Stylesheet Language Transformations (XSLT) library. | Off |

## Disabling an Op Script

You can disable an op script by deleting or deactivating the **file *filename*** statement at the **[edit system scripts op]** hierarchy in the configuration. To determine which op scripts are active on the device, issue the **show configuration system scripts op** operational mode command. The command output lists the enabled op scripts.

To delete an op script from the configuration, perform the following steps:

1. From configuration mode in the CLI, enter the following command:

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

2. Commit the configuration:

   ```
   user@host# commit
   ```

The **file** statement is removed from the configuration for the specified op script, and the **op** operational mode command no longer lists the op script filename as a valid completion.

To deactivate an op script in the configuration, perform the following steps:

1. From configuration mode in the CLI, enter the following command:

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

2. Commit the configuration:

   ```
   user@host# commit
   ```

The filename of the deactivated script remains in the configuration, but it is flagged with **inactive**. For example:

```
[edit system scripts op]
user@host# show

inactive: file script1.xsl;
file script2.xsl;
file script3.xsl;
```

NOTE: You can reactivate an op script using the **activate system scripts op file** *filename* command.

Alternatively, you can delete the script from the **/var/db/scripts/op** directory on a device's hard drive or from the **/config/scripts/op** directory on the flash drive. Only users in the Junos OS **super-user** login class can access and edit files in these directories. If you delete a script, you should also remove the **file** statement at the **[edit system scripts op]** hierarchy level in the configuration. If you delete an op script, but the **file** statement remains in the configuration, the CLI lists this script as a valid completion for the **op** command, but Junos OS issues an invalid filename error when the script is executed.

If you deactivate or delete the **file** statement for an op script in the configuration, you must enable the script again in order to execute it.

**Related Documentation**

- Enabling an Op Script and Defining a Script Alias on page 474

CHAPTER 25

# Troubleshooting Op Scripts



## SLAX Debugger, Profiler, and callflow



### SLAX Debugger, Profiler, and callflow Overview

The Junos OS command-line interface (CLI) and the libslax distribution include the SLAX debugger (sdb), which is used to trace the execution of SLAX scripts. The SLAX debugger enables you to step through script execution, pause script execution at defined breakpoints, and review the value of script variables at any point.

The SLAX debugger operation and command syntax resemble that of the GNU Project Debugger (GDB). Many of the sdb commands follow their GDB counterparts, to the extent possible.Table 16 on page 89 lists the SLAX debugger commands and a brief description of each command.

The SLAX debugger includes a profiler that can report information about the activity and performance of a script. The profiler, which is automatically enabled when you start the debugger, tracks script execution until the script terminates. At any point, profiling information can be displayed or cleared, and the profiler can be temporarily disabled or enabled. The SLAX debugger **callflow** command enables printing of informational data when you enter or exit levels of the script.

#### Table 36: SLAX Debugger Commands

| Command | Description |
|---|---|
| break [*loc*] | Add a breakpoint to the script at the current line of execution. Optionally specify [*file*:]*line* or a template name to create a breakpoint at that position. |
| callflow [on \| off] | Enable or disable callflow tracing. You can explicitly specify the **on** or **off** value. Omitting the value toggles callflow on and off. |

Table 36: SLAX Debugger Commands *(continued)*

| Command | Description |
|---|---|
| continue [*loc*] | Continue running the script until it reaches the next breakpoint. If there are no defined breakpoints, the script runs in its entirety. Optionally, specify [*file*:]*line* or a template name. When you include the optional argument, script execution continues until it reaches either a breakpoint or the specified line number or template name, whichever comes first. |
| delete [*num*] | Delete one or all breakpoints. Breakpoints are numbered sequentially as they are created. Omit the optional argument to delete all breakpoints. Include the breakpoint number as an argument to delete only the specified breakpoint. View currently active breakpoints with the **info** command. |
| finish | Finish executing the current template. |
| help | Display the help message. |
| info [breakpoints \| profile \| profile brief] | Display information about the current script. The default command lists all breakpoints in the script. Optionally specify the **profile** or **profile brief** arguments to display profiling information. |
| list [*loc*] | List the contents of the current script. Optionally specify [*file*:]*line* or a template name from which point the debugger lists partial script contents. The output includes the filename, line number, and code. |
| next | Execute the next instruction, stepping over any function or template calls. |
| over | Execute the next instruction, stepping over any function or template calls or instruction hierarchies. |
| print <*xpath*> | Print the value of the XPath expression. |
| profile [clear \| on \| off \| report \| report brief] | Enable or disable the profiler. The profiler is enabled by default. Include the **clear** option to clear profiling information. Include the **report** or **report brief** option to display profiling information for the current script. |
| quit | Exit debugging mode. |
| reload | Reload the script. |
| run | Restart script execution from the beginning of the script. |
| step | Execute the next instruction, stepping into any function or template calls or instruction hierarchies. |
| where | Show the backtrace of template calls. |

## Using the SLAX Debugger, Profiler, and callflow


- Invoking the SLAX Debugger on page 485
- Using the SLAX Debugger (sdb) on page 485

## Invoking the SLAX Debugger

Both the Junos OS CLI and the SLAX processor in the libslax distribution include the SLAX debugger (sdb), which is used to trace the execution of SLAX scripts.

When you invoke the SLAX debugger, the command-line prompt changes to (sdb) to indicate that you are in debugging mode. For example:

```
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

When using the SLAX debugger from the Junos OS CLI, you can only use the debugger with op scripts that are enabled in the configuration. To invoke the SLAX debugger from the CLI on a device running Junos OS, issue the **op invoke-debugger cli** operational mode command, include the op script name, and optionally include any necessary script arguments.

user@host>**op invoke-debugger cli** ***script*** **<*argument-name argument-value*>**

The following example invokes the SLAX debugger for the op script **ge-interfaces.slax**, which has two parameters, **interface** and **protocol**. Values are supplied for both arguments.

```
user@host> op invoke-debugger cli ge-interfaces interface ge-0/2/0.0 protocol inet
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

To invoke the SLAX debugger when using the SLAX processor, issue the **slaxproc** command with the **--debug** or **-d** option. Specify the script file and any input or output files. If no input file is required, use the **-E** option to indicate an empty input document. If the **-i** or **--input** argument has the value "-", or if you do not include the input option or an input file, standard input is used. When using standard input, press Ctrl+d to signal the end-of-file. The general syntax is:

$ slaxproc --debug [options] [script] [files]

The following example invokes the SLAX debugger for the script **script1.slax** with an empty input document and an output file **script1-output.xml**

```
$ slaxproc --debug -n script1.slax -o script1-output.xml -E
sdb: The SLAX Debugger (version )
Type 'help' for help
(sdb)
```

## Using the SLAX Debugger (sdb)

To view the SLAX debugger help message, issue the **help** command at the (sdb) prompt. To display the help message for a single command, issue **help** ***command***, where *command* is the sdb command for which you want more information. For example:

```
(sdb) help break
 break [loc]    Add a breakpoint at [file:]line or template
```

The process for debugging a script varies depending on the script. A generic outline is presented here:

1. Enter debugging mode.
2. Insert breakpoints in the script using the **break** command.

   During execution, the debugger pauses at defined breakpoints.

   The breakpoint location can be the name of a template or a line number in the current script, or the filename and a line number separated by a colon. If you do not include an argument, a breakpoint is created at the current line of execution. Breakpoints are numbered sequentially as you create them. To view a list of breakpoints, issue the **info breakpoints** command. To delete a breakpoint, issue the **delete** *num* command, and specify the breakpoint number. To delete all breakpoints, issue the **delete** command with no argument.

   The following example creates three breakpoints, the first at line 7, the second at line 25, and the third at the template named "three":

   ```
   (sdb) break 7
   Breakpoint 1 at file script1.slax, line 7
   (sdb) break 25
   Breakpoint 2 at file script1.slax, line 25
   (sdb) break three
   Breakpoint 3 at file script1.slax, line 51
   (sdb) info breakpoints
   List of breakpoints:
     #1 [global] at script1.slax:7
     #2 template two at script1.slax:25
     #3 template three at script1.slax:51
   ```

3. Increment script execution by issuing the **continue**, **finish**, **next**, **over**, and **step** commands at the debugger prompt.

   For example:

   ```
   (sdb) next
   Reached breakpoint 1, at script1.slax:7
   script1.slax:3: var $byte = "10011001";
   ```

4. Review the value of variables as the program executes to ensure that they have the expected value.

   ```
   print xpath-expression
   ```

5. To reload the script contents at any point and restart script execution from the beginning, issue the **reload** command.

   ```
   (sdb) reload
   The script being debugged has been started already.
   Reload and restart it from the beginning? (y or n) y
   Reloading script...
   Reloading complete.
   ```

## Using the SLAX Profiler

The SLAX debugger includes a profiler that can report information about the activity and performance of a script. The profiler, which is automatically enabled when you start the debugger, tracks script execution until the script terminates. At any point, profiling information can be displayed or cleared, and the profiler can be temporarily disabled or enabled.

To access the profiler, issue the **profile** command at the SLAX debugger prompt, (sdb), and include any options. The profile command syntax is:

```
(sdb) profile [options]
```

Table 17 on page 92 lists the profile command options. Issuing the **profile** command with no additional options toggles the profiler on and off.

```
(sdb) profile
Disabling profiler
(sdb)
```

You can access the profiler help by issuing the **help profile** command at the (sdb) prompt.

### Table 37: Profile command options

| Option | Description |
|---|---|
| clear | Clear profiling information |
| off | Disable profiling |
| on | Enable profiling |
| report [brief] | Report profiling information |

To enable the profiler and print a report:

1. Enter debugging mode. The profiler is enabled by default.
2. Step through script execution, or execute a script in its entirety.

   ```
   (sdb) run
   <?xml version="1.0"?>
   <message>Down rev PIC in Fruvenator, Fru-Master 3000</message>
   Script exited normally.
   ```

3. At any point during script execution, display profiling information.

   The **brief** option instructs sdb to avoid showing lines that were not hit, since there is no valid information. If you omit the **brief** option, dashes are displayed.

   ```
   (sdb) profile report brief
   ```

   The following sample output shows a profile report with and without the **brief** option. The source code data in the example is truncated for display purposes.

   ```
   (sdb) profile report
   ```

```
 Line   Hits   User    U/Hit  System    S/Hit Source
    1      -      -        -       -        - version 1.0;
    2      -      -        -       -        -
    3      2      4     2.00       8     4.00 match / {
    4      1     25    25.00      13    13.00     var ....
    5      -      -        -       -        -
    6      -      -        -       -        -     for-each....
    7      1     45    45.00      10    10.00          ..
    8      1     12    12.00       5     5.00         <message>
    9      1     45    45.00      15    15.00          ....
   10      -      -        -       -        -     }
   11      -      -        -       -        - }
Total      6    131              51   Total

(sdb) pro rep b
 Line   Hits   User    U/Hit  System    S/Hit Source
    3      2      4     2.00       8     4.00 match / {
    4      1     25    25.00      13    13.00     var  ....
    7      1     45    45.00      10    10.00          ....
    8      1     12    12.00       5     5.00      <message>
    9      1     45    45.00      15    15.00          ....
Total      6    131              51   Total
```

The profile report includes the following information:

- **Line**—Line number in the source file.
- **Hits**—Number of times this line was executed.
- **User**—Number of microseconds of "user" time spent processing this line.
- **U/Hit**—Average number of microseconds of "user" time per hit.
- **System**—Number of microseconds of "system" time spent processing this line.
- **S/Hit**—Average number of microseconds of "system" time per hit.
- **Source**—Source code line.

This information not only shows how much time is spent during code execution, but can also show which lines are being executed, which can help debug scripts where the execution does not match expectations.

### Using callflow

The SLAX debugger **callflow** command enables printing of informational data when you enter or exit levels of the script.

To enable callflow and view callflow data for a script:

1. Enter debugging mode.
2. Issue the **callflow** command at the SLAX debugger prompt, (sdb).

```
(sdb) callflow
Enabling callflow
```

3. Step through script execution, or execute a script in its entirety.

   Callflow prints information as it enters and exits different levels of the script. Each output line references the instruction, filename, and line number of the frame.

   ```
   (sdb) run
   callflow: 0: enter <xsl:template> in match / at script3.slax:5
   callflow: 1: enter <xsl:template> in template one at script3.slax:14
   callflow: 2: enter <xsl:template> in template two at script3.slax:20
   callflow: 3: enter <xsl:call-template> at script3.slax:22
   ...
   <?xml version="1.0"?>
   <message>Down rev PIC in Fruvenator, Fru-Master 3000</message>
   Script exited normally.
   ```

**Related Documentation**

- op invoke-debugger cli
- Understanding the SLAX Processor (slaxproc) on page 80
- Using the SLAX Processor (slaxproc) on page 84
- libslax Distribution Overview on page 65

CHAPTER 26

# Op Script Examples

This chapter provides sample op scripts that run commands and customize output. This chapter includes the following examples:



## Example: Changing the Configuration Using an Op Script

This example explains how to make structured changes to the Junos OS configuration using an op script.



### Requirements

This example uses a device running Junos OS.

### Overview and Op Script

Op scripts can be used to make structured changes to the Junos OS configuration using the **jcs:load-configuration** template, which is included in the import file **junos.xsl**. Experienced users, who are familiar with Junos OS, can write scripts that prompt for the relevant configuration information and modify the configuration accordingly. This allows users who have less experience with Junos OS to safely modify the configuration using the script.

When called, the **jcs:load-configuration** template performs the following actions:

1. Locks the configuration database
2. Loads the configuration changes
3. Commits the configuration
4. Unlocks the configuration database

The **jcs:load-configuration** template makes changes to the configuration in **configure exclusive** mode. In this mode, Junos OS locks the candidate *global* configuration for as long as the script accesses the shared database and makes changes to the configuration without interference from other users.

If another user is currently editing the configuration in **configure exclusive** mode or if the database is already locked when the template is called, the call fails. In addition, if there are existing, uncommitted changes to the configuration when the template is called, the commit fails. If the template call is successful but the commit fails, Junos OS discards the uncommitted changes and rolls back the configuration.

You provide arguments to the **jcs:load-configuration** template to specify how to integrate the changes into the existing configuration, how to customize the commit operation, what changes to make to the configuration, and which connection handle to use. Starting with Junos OS Release 12.2, you can include the rollback parameter to return the configuration to a previously committed configuration, or you can supply a null configuration for the configuration parameter. If you supply a null configuration to **jcs:load-configuration**, the template performs a simple commit of the candidate configuration. The XSLT and SLAX syntax for the template call is:

```
<xsl:call-template name="jcs:load-configuration">
  <xsl:with-param name="action" select="(merge | override | replace)"/>
  <xsl:with-param name="commit-options" select="node-set"/>
  <xsl:with-param name="configuration" select="configuration-data"/>
  <xsl:with-param name="connection" select="connection-handle"/>
  <xsl:with-param name="rollback" select="number"/>
</xsl:call-template>
```

```
call jcs:load-configuration($action="(merge | override | replace)",
  $commit-options=node-set, $configuration=configuration-data,
  $connection=connection-handle, $rollback=number);
```

The following sample SLAX script demonstrates how to use the **jcs:load-configuration** template to disable an interface on a device running Junos OS. All of the values required for the **jcs:load-configuration** template are defined as variables, which are then passed into the template as arguments.

In this example, the **usage** variable is initialized with a general description of the function of the script. When the script is run, the usage description is output to the CLI using a call to the **jcs:output()** function. This allows the user to verify that he is using the script for the correct purpose.

The script calls the **jcs:get-input()** function and prompts the user to enter the name of the interface that should be disabled. The interface name is stored in the **interface** variable.

The configuration data that includes the changes to the configuration are stored in the variable **config-changes**. This is the value used for the **configuration** parameter of the **jcs:load-configuration** template. This variable includes the Junos XML API tags for the configuration statements that are to be modified. The variable **interface**, which is supplied by the user, designates the name of the interface to disable.

The **load-action** variable is initialized to **merge**, which merges the configuration changes in the **disable** variable with the candidate configuration. This is the equivalent of the CLI configuration mode command **load merge**. Other load options include **replace** and **override**.

The **options** variable uses the **:=** operator to create a node-set, which is passed to the template as the value of the **commit-options** parameter. This example includes the **log** tag to add the description of the commit to the commit log file for future reference.

The call to the **jcs:open()** function opens a connection with the Junos OS management process (mgd) and returns a connection handle that is stored in the **conn_handle** variable. All of the defined variables are passed as arguments to the **jcs:load-configuration** template at the time that it is called.

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";
ns ext = "http://xmlsoft.org/XSLT/namespace";

import "../import/junos.xsl";

match / {
<op-script-results> {

  var $usage = "This script disables the interface specified by the user." _
               "The script modifies the candidate configuration to disable " _
               "the interface and commits the configuration to activate it.";
  var $temp = jcs:output($usage);

  var $interface = jcs:get-input("Enter interface to disable: ");

  var $config-changes = {
    <configuration> {
      <interfaces> {
        <interface> {
          <name> $interface;
          <disable>;
        }
      }
    }
  }

  var $load-action = "merge";

  var $options := {
    <commit-options> {
      <log> "disabling interface " _ $interface;
```

```
    }
  }

  var $conn_handle = jcs:open();

  var $results := {
   call jcs:load-configuration( $action=$load-action, $commit-options=$options,
        $configuration=$config-changes, $connection=$conn_handle);
  }

  $close-results = jcs:close($conn_handle);
}
}
```

The **:=** operator copies the results of the **jcs:load-configuration** template call to a temporary variable and runs the **node-set** function on that variable. The resulting node-set is then stored in the **results** variable. The **:=** operator ensures that the **results** variable is a node-set rather than a result tree fragment so that the script can access the contents. The **jcs:close()** function closes the connection.

By default, the **jcs:load-configuration** template does not output messages to the CLI. To quickly view any issues with the commit, you should add code to the script to output any error or warning messages that are generated as a result of the **jcs:load-configuration** template call:

```
if ($results//xnm:error) {
  for-each ($results//xnm:error) {
    <output> message;
  }
}
if ($results//xnm:warning) {
  for-each ($results//xnm:warning) {
    <output> message;
  }
}
```

## Device Configuration

To download, enable, and test the script:

1. Copy the script into a text file, name the file **config-change.slax**, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **config-change.slax**.

   ```
   [edit system scripts op]
   user@host# set file config-change.slax
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Before running the script, issue the **show interfaces *interface-name*** operational mode command and record the current state of the interface that will be disabled by the script.

5. Execute the op script by issuing the **op config-change** operational mode command.

```
user@host> op config-change
This script disables the interface specified by the user. The script modifies
 the candidate configuration to disable the interface and commits the
configuration to activate it.
Enter interface to disable: so-0/0/0
```

## Verification



### Verifying the Commit

**Purpose** Verify that the commit succeeded.

**Action** You should include code in your script that parses the node-set returned by the **jcs:load-configuration** template for any errors or warnings. This allows you to more easily determine whether the commit succeeded. If there are no warning or error messages, you can verify the success of the commit in several ways.

- Check the commit log to verify that the commit was successful. If you included the **log** option in the **commit-options** parameter, the message should be visible in the commit log along with the commit information.

```
user@host> show system commit
0   2010-09-22 17:08:17 PDT by user via junoscript
    disabling interface so-0/0/0
```

- Check the syslog message file to verify that the commit operation was logged. In this case, you also see an **SNMP_TRAP_LINK_DOWN** message for the disabled interface so-0/0/0. Depending on your configuration settings for traceoptions, this message might or might not appear in your log file.

```
user@host> show log messages | last
Sep 22 17:08:13  host file[7319]: UI_COMMIT: User 'user' requested 'commit'
operation (comment: disabling interface so-0/0/0)
Sep 22 17:08:16  host xntpd[1386]: ntpd exiting on signal 1
Sep 22 17:08:16  host xntpd[1386]: ntpd 4.2.0-a Fri Jun 25 13:48:13 UTC 2010
(1)
Sep 22 17:08:16  host mib2d[1434]: SNMP_TRAP_LINK_DOWN: ifIndex 526,
ifAdminStatus down(2), ifOperStatus down(2), ifName so-0/0/0
```

### Verifying the Configuration Changes

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

**Action**

- Display the configuration and verify that the changes are visible for the specified interface:

```
user@host> show configuration interfaces so-0/0/0
disable;
```

- For this example, you also can issue the **show interfaces *interface-name*** operational mode command to check that the interface was disabled. In this case, the output captured *before* the interface was disabled shows that the interface is **Enabled**:

```
user@host> show interfaces so-0/0/0
Physical interface: so-0/0/0, Enabled, Physical link is Up
  Interface index: 128, SNMP ifIndex: 526
  Link-level type: PPP, MTU: 4474, Clocking: Internal, SONET mode, Speed: OC3,
 Loopback: None, FCS: 16,
  Payload scrambler: Enabled
  Device flags   : Present Running
  Interface flags: Point-To-Point SNMP-Traps Internal: 0x4000
  Link flags     : Keepalives
  CoS queues     : 4 supported, 4 maximum usable queues
  Last flapped   : 2010-09-14 10:33:25 PDT (1w1d 06:27 ago)
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
  SONET alarms   : None
  SONET defects  : None
```

The output captured *after* running the script to disable the interface shows that the interface is now **Administratively down**:

```
user@host> show interfaces so-0/0/0
Physical interface: so-0/0/0, Administratively down, Physical link is Up
  Interface index: 128, SNMP ifIndex: 526
  Link-level type: PPP, MTU: 4474, Clocking: Internal, SONET mode, Speed: OC3,
 Loopback: None, FCS: 16,
  Payload scrambler: Enabled
  Device flags   : Present Running
  Interface flags: Down Point-To-Point SNMP-Traps Internal: 0x4000
  Link flags     : Keepalives
  CoS queues     : 4 supported, 4 maximum usable queues
  Last flapped   : 2010-09-14 10:33:25 PDT (1w1d 06:40 ago)
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
  SONET alarms   : None
  SONET defects  : None
```

**Related Documentation**

- Storing and Enabling Scripts on page 213
- close() Function (jcs Namespace) on page 102
- get-input() Function (jcs and slax Namespaces) on page 109
- jcs:load-configuration Template on page 131
- open() Function (jcs Namespace) on page 112
- output() Function (jcs and slax Namespaces) on page 115

## Example: Customizing Output of the show interfaces terse Command Using an Op Script

This example uses an op script to customize the output of the **show interfaces terse** command. A line-by-line explanation of the XSLT script is provided.



### Requirements

This example uses a device running Junos OS.

### Overview and Op Script

By default, the layout of the **show interfaces terse** command looks like this:

```
user@host> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
dsc                     up    up
fxp0                    up    up
fxp0.0                  up    up   inet     192.168.71.246/21
fxp1                    up    up
fxp1.0                  up    up   inet     10.0.0.4/8
                                   inet6    fe80::200:ff:fe00:4/64
                                            fec0::10:0:0:4/64
                                   tnp      4
gre                     up    up
ipip                    up    up
lo0                     up    up
lo0.0                   up    up   inet     127.0.0.1           --> 0/0
lo0.16385               up    up   inet
                                   inet6    fe80::2a0:a5ff:fe12:2f04
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
tap                     up    up
```

In Junos XML, the output fields are represented as follows:

```
user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
  <interface-information xmlns="http://xml.juniper.net/junos/10.0R1/junos-interface"
 junos:style="terse">
     <physical-interface>
       <name>dsc</name>
       <admin-status>up</admin-status>
       <oper-status>up</oper-status>
       </physical-interface>
       <physical-interface>
         <name>fxp0</name>
         <admin-status>up</admin-status>
```

```
<oper-status>up</oper-status>
<logical-interface>
  <name>fxp0.0</name>
  <admin-status>up</admin-status>
  <oper-status>up</oper-status>
  ... Remainder of output omitted for brevity ...
```

**XSLT Syntax** The following script customizes the output of the **show interfaces terse** command. A line-by-line explanation of the script is provided.

```
<?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="arguments">
    <argument>
      <name>interface</name>
      <description>Name of interface to display</description>
    </argument>
    <argument>
      <name>protocol</name>
      <description>Protocol to display (inet, inet6)</description>
    </argument>
  </xsl:variable>
  <xsl:param name="interface"/>
  <xsl:param name="protocol"/>
  <xsl:template match="/">
    <op-script-results>
      <xsl:variable name="rpc">
        <get-interface-information>
          <terse/>
          <xsl:if test="$interface">
            <interface-name>
              <xsl:value-of select="$interface"/>
            </interface-name>
          </xsl:if>
        </get-interface-information>
      </xsl:variable>
      <xsl:variable name="out" select="jcs:invoke($rpc)"/>
      <interface-information junos:style="terse">
        <xsl:choose>
          <xsl:when test="$protocol='inet' or $protocol='inet6'
              or $protocol='mpls' or $protocol='tnp'">
            <xsl:for-each select="$out/physical-interface/
    logical-interface[address-family/address-family-name = $protocol]">
              <xsl:call-template name="intf"/>
            </xsl:for-each>
          </xsl:when>
          <xsl:when test="$protocol">
            <xnm:error>
              <message>
                <xsl:text>invalid protocol: </xsl:text>
```

```
                    <xsl:value-of select="$protocol"/>
                  </message>
                </xnm:error>
              </xsl:when>
              <xsl:otherwise>
                <xsl:for-each select="$out/physical-interface/logical-interface">
                  <xsl:call-template name="intf"/>
                </xsl:for-each>
              </xsl:otherwise>
            </xsl:choose>
          </interface-information>
        </op-script-results>
      </xsl:template>
      <xsl:template name="intf">
        <xsl:variable name="status">
          <xsl:choose>
            <xsl:when test="admin-status='up' and oper-status='up'">
              <xsl:text> </xsl:text>
            </xsl:when>
            <xsl:when test="admin-status='down'">
              <xsl:text>offline</xsl:text>
            </xsl:when>
            <xsl:when test="oper-status='down' and ../admin-status='down'">
              <xsl:text>p-offline</xsl:text>
            </xsl:when>
            <xsl:when test="oper-status='down' and ../oper-status='down'">
              <xsl:text>p-down</xsl:text>
            </xsl:when>
            <xsl:when test="oper-status='down'">
              <xsl:text>down</xsl:text>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="concat(oper-status, '/', admin-status)"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <xsl:variable name="desc">
          <xsl:choose>
            <xsl:when test="description">
              <xsl:value-of select="description"/>
            </xsl:when>
            <xsl:when test="../description">
              <xsl:value-of select="../description"/>
            </xsl:when>
          </xsl:choose>
        </xsl:variable>
        <logical-interface>
          <name><xsl:value-of select="name"/></name>
          <xsl:if test="string-length($desc)">
            <admin-status><xsl:value-of select="$desc"/></admin-status>
          </xsl:if>
          <admin-status><xsl:value-of select="$status"/></admin-status>
          <xsl:choose>
            <xsl:when test="$protocol">
              <xsl:copy-of
              select="address-family[address-family-name = $protocol]"/>
```

```
 99            </xsl:when>
100            <xsl:otherwise>
101              <xsl:copy-of select="address-family"/>
102            </xsl:otherwise>
103          </xsl:choose>
104        </logical-interface>
105      </xsl:template>
106    </xsl:stylesheet>
```

**Line-by-Line Explanation**

Lines 1 through 7, Line 20, and Lines 105 and 106 are the boilerplate that you include in every op script. For more information, see "Required Boilerplate for Op Scripts" on page 463.

```
 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"/>
     ...
20     <xsl:template match="/">
       ...
105    </xsl:template>
106   </xsl:stylesheet>
```

Lines 8 through 17 declare a variable called **arguments**, containing two arguments to the script: **interface** and **protocol**. This variable declaration causes **interface** and **protocol** to appear in the command-line interface (CLI) as available arguments to the script.

```
 8     <xsl:variable name="arguments">
 9       <argument>
10         <name>interface</name>
11         <description>Name of interface to display</description>
12       </argument>
13       <argument>
14         <name>protocol</name>
15         <description>Protocol to display (inet, inet6)</description>
16       </argument>
17     </xsl:variable>
```

Lines 18 and 19 declare two parameters to the script, corresponding to the arguments created in Lines 8 through 17. The parameter names must exactly match the argument names.

```
18     <xsl:param name="interface"/>
19     <xsl:param name="protocol"/>
```

Lines 20 through 31 declare a variable named **rpc**. The **show interfaces terse** command is assigned to the **rpc** variable. If you include the **interface** argument when you execute the script, the value of the argument (the interface name) is passed into the script.

```
20     <xsl:template match="/">
21       <op-script-results>
22         <xsl:variable name="rpc">
23           <get-interface-information>
24             <terse/>
25             <xsl:if test="$interface">
```

```
26              <interface-name>
27                <xsl:value-of select="$interface"/>
28              </interface-name>
29            </xsl:if>
30          </get-interface-information>
31        </xsl:variable>
```

Line 32 declares a variable named **out** and applies to it the execution of the **rpc** variable (**show interfaces terse** command).

```
32        <xsl:variable name="out" select="jcs:invoke($rpc)"/>
```

Line 33 specifies that the output level of the **show interfaces** command being modified is **terse** (as opposed to **extensive**, **detail**, and so on).

```
33        <interface-information junos:style="terse">
```

Lines 34 through 39 specify that if you include the **protocol** argument when you execute the script and if the protocol value that you specify is **inet**, **inet6**, **mpls**, or **tnp**, the **intf** template is applied to each instance of that protocol type in the output.

```
34          <xsl:choose>
35            <xsl:when test="$protocol='inet' or $protocol='inet6'
              or $protocol='mpls' or $protocol='tnp'">
36              <xsl:for-each select="$out/physical-interface/
          logical-interface[address-family/address-family-name = $protocol]">
37                <xsl:call-template name="intf"/>
38              </xsl:for-each>
39            </xsl:when>
```

Lines 40 through 47 specify that if you include the **protocol** argument when you execute the script and if the protocol value that you specify is something other than **inet**, **inet6**, **mpls**, or **tnp**, an error message is generated.

```
40            <xsl:when test="$protocol">
41              <xnm:error>
42                <message>
43                  <xsl:text>invalid protocol: </xsl:text>
44                  <xsl:value-of select="$protocol"/>
45                </message>
46              </xnm:error>
47            </xsl:when>
```

Lines 48 through 52 specify that if you do not include the **protocol** argument when you execute the script, the **intf** template is applied to each logical interface in the output.

```
48            <xsl:otherwise>
49              <xsl:for-each select="$out/physical-interface/logical-interface">
50                <xsl:call-template name="intf"/>
51              </xsl:for-each>
52            </xsl:otherwise>
```

Lines 53 through 56 are closing tags.

```
53          </xsl:choose>
54        </interface-information>
55      </op-script-results>
56    </xsl:template>
```

Line 57 opens the **intf** template. This template customizes the output of the **show interfaces terse** command.

```
57    <xsl:template name="intf">
```

Line 58 declares a variable called **status**, the purpose of which is to specify how the interface status is reported. Lines 59 through 78 contain a **<xsl:choose>** instruction that populates the **status** variable by considering all the possible states. As always in XSLT, the first **<xsl:when>** instruction that evaluates as TRUE is executed, and the remainder are ignored. Each **<xsl:when>** instruction is explained separately.

```
58      <xsl:variable name="status">
59        <xsl:choose>
```

Lines 60 through 62 specify that if **admin-status** is up and **oper-status** is **up**, no output is generated. In this case, the **status** variable remains empty.

```
60          <xsl:when test="admin-status='up' and oper-status='up'">
61            <xsl:text> </xsl:text>
62          </xsl:when>
```

Lines 63 through 65 specify that if **admin-status** is **down**, the **status** variable contains the text **offline**.

```
63          <xsl:when test="admin-status='down'">
64            <xsl:text>offline</xsl:text>
65          </xsl:when>
```

Lines 66 through 68 specify that if **oper-status** is **down** and the physical interface **admin-status** is **down**, the **status** variable contains the text **p-offline**. (**../** selects the physical interface.)

```
66          <xsl:when test="oper-status='down' and ../admin-status='down'">
67            <xsl:text>p-offline</xsl:text>
68          </xsl:when>
```

Lines 69 through 71 specify that if **oper-status** is **down** and the physical interface **oper-status** is **down**, the **status** variable contains the text **p-down**. (**../** selects the physical interface.)

```
69          <xsl:when test="oper-status='down' and ../oper-status='down'">
70            <xsl:text>p-down</xsl:text>
71         </xsl:when>
```

Lines 72 through 74 specify that if **oper-status** is **down**, the **status** variable contains the text **down**.

```
72          <xsl:when test="oper-status='down'">
73            <xsl:text>down</xsl:text>
74          </xsl:when>
```

Lines 75 through 77 specify that if none of the test cases are true, the **status** variable contains **oper-status** and **admin-status** concatenated with a slash as a separator.

```
75          <xsl:otherwise>
76            <xsl:value-of select="concat(oper-status, '/', admin-status)"/>
77          </xsl:otherwise>
```

Lines 78 and 79 are closing tags.

```
78        </xsl:choose>
```

```
79      </xsl:variable>
```

Lines 80 through 89 define a variable called **desc**. An **<xsl:choose>** instruction populates the variable by selecting the most specific interface description available. If a logical interface description is included in the configuration, it is used to populate the **desc** variable. If not, the physical interface description is used. If no physical interface description is included in the configuration, the variable remains empty. As always in XSLT, the first **<xsl:when>** instruction that evaluates as TRUE is executed, and the remainder are ignored.

```
80      <xsl:variable name="desc">
81        <xsl:choose>
82          <xsl:when test="description">
83            <xsl:value-of select="description"/>
84          </xsl:when>
85          <xsl:when test="../description">
86            <xsl:value-of select="../description"/>
87          </xsl:when>
88        </xsl:choose>
89      </xsl:variable>
```

The remainder of the script specifies how the operational mode output is displayed.

Lines 90 and 91 specify that the logical interface name is displayed first in the output.

```
90      <logical-interface>
91        <name><xsl:value-of select="name"/></name>
```

Lines 92 through 94 test whether the **desc** variable has a nonzero number of characters. If the number of characters is more than zero, the interface description is displayed in the standard location of the **admin-status** field. (In standard output, the **admin-status** field is displayed on the second line.)

```
92        <xsl:if test="string-length($desc)">
93          <admin-status><xsl:value-of select="$desc"/></admin-status>
94        </xsl:if>
```

Line 95 specifies that the interface status as defined in the **status** variable is displayed next.

```
95        <admin-status><xsl:value-of select="$status"/></admin-status>
```

Lines 96 through 103 specify that if you include the **protocol** argument when you execute the script, only interfaces with that protocol configured are displayed. If you do not include the **protocol** argument, all interfaces are displayed.

```
96        <xsl:choose>
97          <xsl:when test="$protocol">
98            <xsl:copy-of
              select="address-family[address-family-name = $protocol]"/>
99          </xsl:when>
100         <xsl:otherwise>
101           <xsl:copy-of select="address-family"/>
102         </xsl:otherwise>
103       </xsl:choose>
```

Lines 104 through 106 are closing tags.

```
104     </logical-interface>
105   </xsl:template>
```

```
106   </xsl:stylesheet>
```

**SLAX Syntax** The SLAX version of the script 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";

var $arguments = {
  <argument> {
    <name> "interface";
    <description> "Name of interface to display";
  }
  <argument> {
    <name> "protocol";
    <description> "Protocol to display (inet, inet6)";
  }
}
param $interface;
param $protocol;
match / {
  <op-script-results> {
    var $rpc = {
      <get-interface-information> {
        <terse>;
        if ($interface) {
          <interface-name> $interface;
        }
      }
    }
    var $out = jcs:invoke($rpc);
    <interface-information junos:style="terse"> {
      if ($protocol='inet' or $protocol='inet6' or $protocol='mpls' or
                $protocol='tnp') {
        for-each ($out/physical-interface/
            logical-interface[address-family/address-family-name = $protocol]) {
          call intf();
        }
      } else if ($protocol) {
        <xnm:error> {
          <message> {
            expr "invalid protocol: ";
            expr $protocol;
          }
        }
      } else {
        for-each ($out/physical-interface/logical-interface) {
          call intf();
        }
      }
    }
  }
}
intf () {
```

```
var $status = {
  if (admin-status='up' and oper-status='up') {
  } else if (admin-status='down') {
    expr "offline";
  } else if (oper-status='down' and ../admin-status='down') {
    expr "p-offline";
  } else if (oper-status='down' and ../oper-status='down') {
    expr "p-down";
  } else if (oper-status='down') {
    expr "down";
  } else {
    expr oper-status _ '/' _ admin-status;
  }
}
var $desc = {
  if (description) {
    expr description;
  } else if (../description) {
    expr ../description;
  }
}
<logical-interface> {
  <name> name;
  if (string-length($desc)) {
    <admin-status> $desc;
  }
  <admin-status> $status;
  if ($protocol) {
    copy-of address-family[address-family-name = $protocol];
  } else {
    copy-of address-family;
  }
 }
}
```

## 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 **interface.xsl** or **interface.slax** as appropriate, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **interface.xsl** or **interface.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file interface.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op interface** operational mode command.

## Verification

### Verifying the Commit Script Output

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

**Action** Issue the **show interfaces terse** and **op interface** operational commands and compare the output. The **show interfaces terse** command displays the standard output. The **op interface** command displays the customized output.

```
user@host> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
dsc                     up    up
fxp0                    up    up
fxp0.0                  up    up   inet     192.168.71.246/21
fxp1                    up    up
fxp1.0                  up    up   inet     10.0.0.4/8
                                   inet6    fe80::200:ff:fe00:4/64
                                            fec0::10:0:0:4/64
                                   tnp      4
gre                     up    up
ipip                    up    up
lo0                     up    up
lo0.0                   up    up   inet     127.0.0.1           --> 0/0
lo0.16385               up    up   inet
                                   inet6    fe80::2a0:a5ff:fe12:2f04
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
tap                     up    up

user@host> op interface
Interface               Admin Link Proto    Local                 Remote
fxp0.0                  This is the Ethernet Management interface.
                                   inet     192.168.71.246/21
fxp1.0                             inet     10.0.0.4/8
                                   inet6    fe80::200:ff:fe00:4/64
                                            fec0::10:0:0:4/64
                                   tnp      4
lo0.0                              inet     127.0.0.1           --> 0/0
lo0.16385                          inet
                                   inet6    fe80::2a0:a5ff:fe12:2f04-->
```

Issue the **op interface** operational command for different hierarchy levels and review the output. For example:

```
user@host> op interface interface fxp0
Interface               Admin Link Proto    Local                 Remote
fxp0.0                  This is the Ethernet Management interface.
                                   inet     192.168.71.246/21

user@host> op interface protocol inet
Interface               Admin Link Proto    Local                 Remote
fxp0.0                  This is the Ethernet Management interface.
                                   inet     192.168.71.246/21
fxp1.0                             inet     10.0.0.4/8
lo0.0                              inet     127.0.0.1           --> 0/0
lo0.16385                          inet
```

## Example: Displaying DNS Hostname Information Using an Op Script

This example uses an op script to display Domain Name System (DNS) information for a device in your network.



## Requirements

This example uses a device running Junos OS.

## Overview and Op Script

This script displays DNS information for a device in your network. The script offers a slight improvement over the **show host *hostname*** command because you do not need to enter a hostname or IP address to view DNS information for the device you are currently using.

There is no Junos Extensible Markup Language (XML) equivalent for the **show host *hostname*** command. Therefore, this script uses the **show host *hostname*** command directly rather than using a remote procedure call (RPC).

The script is provided in two distinct versions, one using the **<xsl:choose>** element and the other using the **jcs:first-of()** function. Both versions accept the same argument and produce the same output. Each version is shown in both XSLT and SLAX syntax.

**XSLT Syntax Using the <xsl:choose> Element**

```
<?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="arguments">
    <argument>
      <name>dns</name>
      <description>Name or IP address of a host</description>
    </argument>
  </xsl:variable>
  <xsl:param name="dns"/>
  <xsl:template match="/">
    <op-script-results>
      <xsl:variable name="query">
        <xsl:choose>
          <xsl:when test="$dns">
            <command>
              <xsl:value-of select="concat('show host ', $dns)"/>
            </command>
          </xsl:when>
          <xsl:when test="$hostname">
            <command>
              <xsl:value-of select="concat('show host ', $hostname)"/>
            </command>
```

```
            </xsl:when>
          </xsl:choose>
        </xsl:variable>
        <xsl:variable name="result" select="jcs:invoke($query)"/>
        <xsl:variable name="host" select="$result"/>
        <output>
          <xsl:value-of select="concat('Name: ', $host)"/>
        </output>
      </op-script-results>
    </xsl:template>
</xsl:stylesheet>
```

### XSLT Syntax Using the jcs:first-of() Function

```
<?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="arguments">
    <argument>
      <name>dns</name>
      <description>Name or IP address of a host</description>
    </argument>
  </xsl:variable>
  <xsl:param name="dns"/>
  <xsl:template match="/">
    <op-script-results>
      <xsl:variable name="target" select="jcs:first-of($dns, $hostname)"/>
      <xsl:variable name="query">
        <command>
          <xsl:value-of select="concat('show host ', $target)"/>
        </command>
      </xsl:variable>
      <xsl:variable name="result" select="jcs:invoke($query)"/>
      <xsl:variable name="host" select="$result"/>
      <output>
        <xsl:value-of select="concat('Name: ', $host)"/>
      </output>
    </op-script-results>
  </xsl:template>
</xsl:stylesheet>
```

### SLAX Syntax Using the <xsl:choose> Element

```
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 $arguments = {
  <argument> {
    <name> "dns";
    <description> "Name or IP address of a host";
  }
```

```
}
param $dns;
match / {
  <op-script-results> {
    var $query = {
      if ($dns) {
        <command> 'show host ' _ $dns;
      } else if ($hostname) {
        <command> 'show host ' _ $hostname;
      }
    }
    var $result = jcs:invoke($query);
    var $host = $result;
    <output> 'Name: ' _ $host;
  }
}
```

**SLAX Syntax Using the jcs:first-of() Function**

```
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 $arguments = {
  <argument> {
    <name> "dns";
    <description> "Name or IP address of a host";
  }
}
param $dns;
match / {
  <op-script-results> {
    var $target = jcs:first-of($dns, $hostname);
    var $query = {
      <command> 'show host ' _ $target;
    }
    var $result = jcs:invoke($query);
    var $host = $result;
    <output> 'Name: ' _ $host;
  }
}
```

## 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 **hostname.xsl** or **hostname.slax** as appropriate, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **hostname.xsl** or **hostname.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file hostname.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op hostname <dns (*hostname* | *address*)>** operational mode command.

## Verification

### Verifying the Commit Script Execution

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

**Action** When you issue the **op hostname** operational mode command without the **dns** option, DNS information is displayed for the local device:

```
user@host1> op hostname
Name:
host1 has address 10.168.71.246
```

When you issue the **op hostname dns *hostname*** command, DNS information is displayed for the specified device:

```
user@host1> op hostname dns router1
Name:
router1 has address 10.168.71.249
```

When you issue the **op hostname dns *address*** command, DNS information is displayed for the specified address:

```
user@host1> op hostname dns 10.168.71.249
Name:
249.71.168.10.IN-ADDR.ARPA domain name pointer router1
```

# Example: Exporting Files Using an Op Script

The op script in this example uses the Junos XML protocol **file-put** operation to write to a file on a remote server and on the local device.



## Requirements

This example uses a device running Junos OS.

## Overview and Op Script

The Junos XML protocol **file-put** operation creates a file and writes the specified contents to that file. The basic syntax for using the **file-put** command is as follows:

```
<rpc>
  <file-put>
    <delete-if-exist />
    <encoding>value</encoding>
    <filename>value</filename>
    <permission>value</permission>
    <file-contents>file</file-contents>
  </file-put>
</rpc>
```

The following tag elements are used with the **file-put** command. These tags can be placed in any order with the exception of **file-contents**. The **file-contents** tag element must be the last tag in list.

- **delete-if-exist**—(Optional) If included, any existing file is overwritten. If the tag is omitted, an error is returned if an existing file is encountered.
- **encoding**—(Mandatory) Specifies the type of encoding used. You can use **ASCII** or **base64** encoding.
- **filename**—(Mandatory) Within this tag, you include the full or relative path and filename of the file to create. When you use a relative path, the specified path is relative to the user's home directory. If the specified directory does not exist, the system returns a "directory does not exist" error.
- **permission**—(Optional) Sets the file's UNIX permission on the remote server. For example, to apply read/write access for the user, and read access to others, you would set the permission value to 0644. For a full explanation of UNIX permissions, see the **chmod** command.
- **file-contents**—(Mandatory) The **ASCII** or **base64** encoded file contents to export. This must be the last tag in the list.

**XSLT Syntax**

The following sample script executes a Junos XML API request and exports the results to a file on a remote device and a file on the local device. The script takes three arguments: the IP address or hostname of the remote device, the filename, and the file encoding. The **arguments** variable is declared at the global level of the script so that the argument names and descriptions are visible in the command-line interface (CLI).

The script invokes the Junos XML API **<get-software-information>** request on the local device and stores the result in the **result** variable. The script declares the **fileput** variable, which contains the remote procedure call (RPC) for the **file-put** operation. The command-line arguments define the values for the **filename** and **encoding** tag elements. If the mandatory argument **myhost** is missing, the script issues an error and halts execution. Otherwise, the script prompts for the username and password that will be used to connect to the remote device.

If connection to the remote device is successful, the script executes the RPC within the context of the connection handle. The output of the **file-put** operation, which is the result of the **jcs:execute()** function, is stored in the **out** variable. If the operation encounters an error, the script prints the error to the CLI. If the **file-put** operation is successful, the contents specified by the **file-contents** tag element are exported to the specified file on the remote device. The connection to the remote host is then closed. The script also exports the contents to an identical file on the local device.

The sample script includes the optional tag elements **permission** and **delete-if-exist** for the **file-put** operation. By including the **delete-if-exist** tag, the script overwrites any existing file of the same name on the remote and local hosts. In this example, the **permission** tag is set to **0644**.

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet
   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" version="1.0">
 <xsl:import href="../import/junos.xsl"/>

 <xsl:variable name="arguments">
  <argument>
   <name>myhost</name>
   <description>IP address or hostname of the remote host</description>
  </argument>
  <argument>
   <name>filename</name>
   <description>name of destination file</description>
  </argument>
  <argument>
   <name>encoding</name>
   <description>ascii or base64</description>
  </argument>
 </xsl:variable>

 <xsl:param name="myhost"/>
 <xsl:param name="filename"/>
 <xsl:param name="encoding"/>

 <xsl:template match="/">
  <op-script-results>
```

```
<xsl:variable name="rpc">
 <get-software-information/>
</xsl:variable>
<xsl:variable name="result" select="jcs:invoke($rpc)"/>

<xsl:variable name="fileput">
 <file-put>
  <filename>
   <xsl:value-of select="$filename"/>
  </filename>
  <encoding>
   <xsl:value-of select="$encoding"/>
  </encoding>
  <permission>0644</permission>
  <delete-if-exist/>
  <file-contents>
   <xsl:value-of select="$result"/>
  </file-contents>
 </file-put>
</xsl:variable>

<xsl:choose>
 <xsl:when test="$myhost = ''">
  <xnm:error>
   <message>missing mandatory argument 'myhost'</message>
  </xnm:error>
 </xsl:when>
 <xsl:otherwise>
  <xsl:variable name="username" select="jcs:get-input('Enter username: ')"/>
  <xsl:variable name="pw" select="jcs:get-secret('Enter password: ')"/>
  <xsl:variable name="connect" select="jcs:open($myhost, $username, $pw)"/>
  <xsl:choose>
   <xsl:when test="$connect">
    <output>Connected to host.  Exporting file... </output>
    <xsl:variable name="out" select="jcs:execute($connect, $fileput)"/>
    <xsl:choose>
     <xsl:when test="$out//xnm:error">
      <xsl:copy-of select="($out//xnm:error)"/>
     </xsl:when>
     <xsl:otherwise>
      <output>
       <xsl:value-of select="$out"/>
      </output>
     </xsl:otherwise>
    </xsl:choose>
    <xsl:value-of select="jcs:close($connect)"/>
   </xsl:when>
   <xsl:otherwise>
    <output>No connection to host.</output>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:otherwise>
</xsl:choose>

<xsl:variable name="local-out" select="jcs:invoke($fileput)"/>
```

```
    <output>
     <xsl:value-of select="concat('Saving file on local host\n', $local-out)"/>
    </output>
   </op-script-results>
 </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 $arguments = {
   <argument> {
      <name> "myhost";
      <description> "IP address or hostname of the remote host";
   }
   <argument> {
      <name> "filename";
      <description> "name of destination file";
   }
   <argument> {
      <name> "encoding";
      <description> "ascii or base64";
   }
}

param $myhost;
param $filename;
param $encoding;

match / {
   <op-script-results> {

      var $rpc = <get-software-information>;
      var $result = jcs:invoke($rpc);

      var $fileput = {
         <file-put> {
            <filename>$filename;
            <encoding>$encoding;
            <permission>'0644';
            <delete-if-exist>;
            <file-contents>$result;
         }
      }

      if ($myhost = '') {
         <xnm:error> {
            <message> "missing mandatory argument 'myhost'";
         }
      }
      else {
         var $username = jcs:get-input("Enter username: ");
         var $pw = jcs:get-secret("Enter password: ");
```

```
        var $connect = jcs:open($myhost, $username, $pw);

        if ($connect) {
           <output> "Connected to host.  Exporting file... \n";
           var $out = jcs:execute($connect, $fileput);
           if ($out//xnm:error) {
              copy-of ($out//xnm:error);
           }
           else {
              <output> $out;
           }
           expr jcs:close($connect);

        }
        else {
           <output> "No connection to host.";
        }

      }
      var $local-out = jcs:invoke($fileput);
      <output> "Saving file on local host\n" _ $local-out;
   }
}
```

## 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 **export.xsl** or **export.slax** as appropriate, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **export.xsl** or **export.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file export.(slax | xsl)
   ```

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

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

4. Execute the op script by issuing the **op export** operational mode command and include any necessary arguments.

## Verification


- Verifying the Op Script Arguments on page 516
- Verifying Op Script Execution on page 517


### Verifying the Op Script Arguments

**Purpose** Verify that the argument names and descriptions show up in the CLI.

**Action** Issue the **op exort ?** operational mode command. The CLI lists the possible completions for the script arguments based on the definitions within the global **arguments** variable in the script.

```
user@host> op export ?
Possible completions:
  <[Enter]>            Execute this command
  <name>               Argument name
  detail               Display detailed output
  encoding             ascii or base64
  filename             name of destination file
  myhost               IP address or hostname of the remote host
  |                    Pipe through a command
```

### Verifying Op Script Execution

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

**Action** Issue the **op export myhost *host* encoding *encoding* filename *file*** operational mode command, and include the appropriate username and password when prompted. If script execution is successful, the result of the **<get-software-information>** RPC request is written to the file on the remote device and also on the local device. For example:

```
root@host> op export myhost router1 encoding ascii filename /var/log/host-version.txt
Enter username: root
Enter password:
Connected to host.  Exporting file...

/var/log/host-version.txt
Saving file on local host

/var/log/host-version.txt
```

If you fail to supply the IP address or hostname of the remote device in the command-line arguments, the script issues an error and halts execution.

```
root@host> op export
error: missing mandatory argument 'myhost'
```

If you omit the **delete-if-exist** child tag of the **file-put** operation, and the specified file already exists, the script reports an error.

```
root@host> op export myhost router1 encoding ascii filename /var/log/host-version.txt
Enter username: root
Enter password:
Connected to host.  Exporting file...

Destination file exists
Saving file on local host

Destination file exists
```

If you execute the script and include a directory path that does not exist on either the remote or the local host, the script reports an error.

```
root@host> op export myhost router1 encoding ascii filename /var/test/host-version.txt
Enter username: root
Enter password:
Connected to host.  Exporting file...

Destination directory does not exist: /var/test
Saving file on local host

Destination directory does not exist: /var/test
```

**Related Documentation**

- Declaring Arguments in Op Scripts on page 469
- Example: Importing Files Using an Op Script on page 523

## Example: Finding LSPs to Multiple Destinations Using an Op Script

This example uses an op script to check for label-switched paths (LSPs) to multiple destinations.



### Requirements

This example uses a device running Junos OS.

### Overview and Op Script

The following example script, which is shown in both XSLT and SLAX, checks for LSPs to multiple destinations. The script takes one mandatory command-line argument, the address specifying the LSP endpoint. The address argument can include an optional prefix length. If no address is specified, the script generates an error message and halts execution.

The **get-configuration** variable stores the remote procedure call (RPC) to retrieve the **[edit protocols mpls]** hierarchy level of the device's committed configuration. This configuration is stored in the **config** variable. The **get-route-information** variable stores the RPC equivalent of the **show route *address* terse** operational mode command, where the value of the **destination** tag specifies *address*. The script sets this value to the address specified by the user on the command line. The script invokes the **get-route-information** RPC and stores the output in the **rpc-out** variable. If **rpc-out** does not contain any errors, the script examines all host route entries present at the **route-table/rt/rt-destination** node.

For each host route entry, if an LSP to the destination is configured in the active configuration, the script generates a "Found" message with the destination address and corresponding LSP name in the output. If an LSP to the destination is not configured, the output generates a "Missing" message containing the destination address and hostname.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet
  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" version="1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:variable name="arguments">
    <argument>
      <name>address</name>
      <description>LSP endpoint</description>
    </argument>
  </xsl:variable>
```

```
<xsl:param name="address"/>
<xsl:template match="/">
  <op-script-output>
    <xsl:choose>
      <xsl:when test="$address = ''">
        <xnm:error>
          <message>missing mandatory argument 'address'</message>
        </xnm:error>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="get-configuration">
          <get-configuration database="committed">
            <configuration>
              <protocols>
                <mpls/>
              </protocols>
            </configuration>
          </get-configuration>
        </xsl:variable>
        <xsl:variable name="config"
              select="jcs:invoke($get-configuration)"/>
        <xsl:variable name="mpls" select="$config/protocols/mpls"/>
        <xsl:variable name="get-route-information">
          <get-route-information>
            <terse/>
            <destination>
              <xsl:value-of select="$address"/>
            </destination>
          </get-route-information>
        </xsl:variable>
        <xsl:variable name="rpc-out"
                select="jcs:invoke($get-route-information)"/>
        <xsl:choose>
          <xsl:when test="$rpc-out//xnm:error">
            <xsl:copy-of select="$rpc-out//xnm:error"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:for-each select="$rpc-out/route-table/rt/rt-destination">
              <xsl:choose>
                <xsl:when test="contains(.,'/32')">
                  <xsl:variable name="dest"
                        select="substring-before(.,'/')"/>
                  <xsl:variable name="lsp"
                      select="$mpls/label-switched-path[to = $dest]"/>
                  <xsl:choose>
                    <xsl:when test="$lsp">
                      <output>
                       <xsl:value-of select="concat('Found: ', $dest,
                                ' (',$lsp/to, ') --&gt; ', $lsp/name)"/>
                      </output>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:variable name="name"
                                  select="jcs:hostname($dest)"/>
                      <output>
                        <xsl:value-of select="concat('Name: ', $name)"/>
```

```
                                    </output>
                                    <output>
                                      <xsl:value-of select="concat('Missing: ', $dest)"/>
                                    </output>
                                  </xsl:otherwise>
                                </xsl:choose>
                              </xsl:when>
                              <xsl:otherwise>
                                <output>
                                  <xsl:value-of select="concat('Not a host route: ', .)"/>
                                </output>
                              </xsl:otherwise>
                            </xsl:choose>
                          </xsl:for-each>
                        </xsl:otherwise>
                      </xsl:choose>
                    </xsl:otherwise>
                  </xsl:choose>
                </op-script-output>
              </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 $arguments = {
  <argument> {
    <name> "address";
    <description> "LSP endpoint";
  }
}
param $address;
match / {
  <op-script-output> {
    if ($address = '') {
      <xnm:error> {
        <message> "missing mandatory argument 'address'";
      }
    } else {
      var $get-configuration = {
        <get-configuration database="committed"> {
          <configuration> {
            <protocols> {
              <mpls>;
            }
          }
        }
      }
      var $config = jcs:invoke($get-configuration);
      var $mpls = $config/protocols/mpls;
      var $get-route-information = {
        <get-route-information> {
```

```
        <terse>;
        <destination> $address;
      }
    }
    var $rpc-out = jcs:invoke($get-route-information);
    if ($rpc-out//xnm:error) {
      copy-of $rpc-out//xnm:error;
    } else {
      for-each ($rpc-out/route-table/rt/rt-destination) {
        if (contains(.,'/32')) {
          var $dest = substring-before(.,'/');
          var $lsp = $mpls/label-switched-path[to = $dest];
          if ($lsp) {
            <output> 'Found: ' _ $dest _ ' (' _ $lsp/to _ ') --> ' _
                $lsp/name;
          } else {
            var $name = jcs:hostname($dest);
            <output> 'Name: ' _ $name;
            <output> 'Missing: ' _ $dest;
          }
        } else {
          <output> 'Not a host route: ' _ .;
        }
      }
    }
  }
}
```

## 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.xsl** or **lsp.slax** as appropriate, and copy it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **lsp.xsl** or **lsp.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file lsp.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op lsp address** ***address*** operational mode command.

## Verification

### Verifying Script Execution

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

**Action** Issue the **op lsp address *address*** operational mode command to execute the script. The output varies depending on the configuration.

```
user@R4> op lsp address 10.168.215.0/24
Found: 192.168.215.1 (192.168.215.1) --> R4>R1
Found: 192.168.215.2 (192.168.215.2) --> R4>R2
Name: R3
Missing: 10.168.215.3
Name: R5
Missing: 10.168.215.4
Name: R6
Missing: 10.168.215.5
```

## Example: Importing Files Using an Op Script

The op script in this example uses the Junos XML protocol **file-get** operation to read the contents of a file from a remote server.



### Requirements

This example uses a device running Junos OS.

### Overview and Op Script

The Junos XML protocol **file-get** operation reads the contents of a file. The basic syntax for using the **file-get** command is as follows:

```
<rpc>
  <file-get>
    <filename>value</filename>
    <encoding>value</encoding>
  </file-get>
</rpc>
```

The following tag elements are used with the **file-get** command.

- **encoding**—(Mandatory) Specifies the type of encoding used. You can use **ASCII**, **base64**, or **raw** encoding.
- **filename**—(Mandatory) Within this tag, you include the full or relative path and filename of the file to import. When you use a relative path, the specified path is relative to the **/var/tmp/** directory if the **file-get** operation is executed locally. If the operation is executed remotely within the context of a connection handle, the path is relative to the user's home directory.

NOTE: When you use ASCII encoding, the file-get operation converts any control characters in the imported file to the Unicode character 'SECTION SIGN' (U+00A7).

**XSLT Syntax**

The following sample script connects to a remote device and reads the contents of the specified file. The script takes three arguments: the IP address or hostname of the remote device, the filename, and the file encoding. The **arguments** variable is declared at the global level of the script so that the argument names and descriptions are visible in the command-line interface (CLI).

The script declares the **fileget** variable, which contains the remote procedure call (RPC) for the **file-get** operation. The command-line arguments define the values for the **filename** and **encoding** tag elements. If the mandatory argument **myhost** is missing, the script issues an error and halts execution. Otherwise, the script prompts for the username and password that will be used to connect to the remote device.

If connection to the remote device is successful, the script executes the RPC within the context of the connection handle. The output of the **file-get** operation, which is the result of the **jcs:execute()** function, is stored in the **out** variable. If the operation encounters an error, the script prints the error to the CLI. If the **file-get** operation is successful, the contents of the file are stored in the **out** variable, which is printed to the CLI. The connection to the remote host is then closed.

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet
   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" version="1.0">
  <xsl:import href="../import/junos.xsl"/>

 <xsl:variable name="arguments">
   <argument>
      <name>myhost</name>
      <description>IP address or hostname of the remote host</description>
   </argument>
   <argument>
      <name>filename</name>
      <description>name of file</description>
   </argument>
   <argument>
      <name>encoding</name>
      <description>ascii, base64, or raw</description>
   </argument>
 </xsl:variable>

 <xsl:param name="myhost"/>
 <xsl:param name="filename"/>
 <xsl:param name="encoding"/>

 <xsl:template match="/">
   <op-script-results>
      <xsl:variable name="fileget">
         <file-get>
```

```
            <filename>
              <xsl:value-of select="$filename"/>
            </filename>
            <encoding>
              <xsl:value-of select="$encoding"/>
            </encoding>
          </file-get>
        </xsl:variable>
        <xsl:choose>
          <xsl:when test="$myhost = ''">
            <xnm:error>
              <message>missing mandatory argument 'myhost'</message>
            </xnm:error>
          </xsl:when>
          <xsl:otherwise>
            <xsl:variable name="username" select="jcs:get-input('Enter username: ')"/>

            <xsl:variable name="pw" select="jcs:get-secret('Enter password: ')"/>
           <xsl:variable name="connect" select="jcs:open($myhost, $username, $pw)"/>

            <xsl:choose>
              <xsl:when test="$connect">
                <output>Connected to host.  Reading file...
                </output>
                <xsl:variable name="out" select="jcs:execute($connect, $fileget)"/>
                <xsl:choose>
                  <xsl:when test="$out//xnm:error">
                    <xsl:copy-of select="$out//xnm:error"/>
                  </xsl:when>
                  <xsl:otherwise>
                    <output>
                      <xsl:value-of select="concat('File contents: ', $out)"/>
                    </output>
                  </xsl:otherwise>
                </xsl:choose>
                <xsl:value-of select="jcs:close($connect)"/>
              </xsl:when>
              <xsl:otherwise>
                <output>No connection to host.</output>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:otherwise>
        </xsl:choose>
      </op-script-results>
    </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 $arguments = {
  <argument> {
    <name> "myhost";
```

```
    <description> "IP address or hostname of the remote host";
  }
  <argument> {
    <name> "filename";
    <description> "name of file";
  }
  <argument> {
    <name> "encoding";
    <description> "ascii, base64, or raw";
  }
}

param $myhost;
param $filename;
param $encoding;

match / {
  <op-script-results> {
    var $fileget = {
      <file-get> {
        <filename>$filename;
        <encoding>$encoding;
      }
    }

    if ($myhost = '') {
      <xnm:error> {
        <message> "missing mandatory argument 'myhost'";
      }
    }
    else {
      var $username = jcs:get-input("Enter username: ");
      var $pw = jcs:get-secret("Enter password: ");
      var $connect = jcs:open($myhost, $username, $pw);

      if ($connect) {
        <output> "Connected to host.  Reading file... \n";
        var $out = jcs:execute($connect, $fileget);
        if ($out//xnm:error) {
          copy-of $out//xnm:error;
        }
        else {
          <output> "File contents: " _ $out;
        }
        expr jcs:close($connect);
      }
      else {
        <output> "No connection to host.";
      }
    }
  }
}
```

## 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/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **import.xsl** or **import.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file import.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op import** operational mode command and include any necessary arguments.

## Verification

### Verifying the Script Arguments

**Purpose** Verify that the argument names and descriptions show up in the CLI.

**Action** Issue the **op import ?** operational mode command. The CLI lists the possible completions for the script arguments based on the definitions within the global **arguments** variable in the script.

```
user@host> op import ?
Possible completions:
  <[Enter]>            Execute this command
  <name>               Argument name
  detail               Display detailed output
  encoding             ascii, base64, or raw
  filename             name of file
  myhost               IP address or hostname of the remote host
  |                    Pipe through a command
```

### Verifying Op Script Execution

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

**Action** Issue the **op import myhost *host* encoding *encoding* filename *file*** operational mode command, and include the appropriate username and password when prompted. If script execution is successful, the contents of the requested file are displayed. For example:

```
root@host> op import myhost router1 encoding ascii filename /var/db/scripts/op/test.slax
Enter username: root
Enter password:
Connected to host.  Reading file...
File contents:

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";
...
```

If you fail to supply the IP address or hostname of the remote device in the command-line arguments, the script issues an error and halts execution.

```
root@host> op import
error: missing mandatory argument 'myhost'
```

Also, if the specified path or file does not exist, the script issues an error.

```
root@host> op import myhost router1 encoding ascii filename /var/db/scripts/op/test1.slax
Enter username: root
Enter password:
Connected to host.  Reading file...
File contents:

Failed to open file (/var/db/scripts/op/test1.slax): No such file or directory
```

**Related Documentation**

- Declaring Arguments in Op Scripts on page 469
- Example: Exporting Files Using an Op Script on page 511

## Example: Restarting an FPC Using an Op Script

This example uses an op script to restart a Flexible PIC Concentrator (FPC).



### Requirements

This example uses a device running Junos OS that contains a Flexible PIC Concentrator (FPC) or equivalent component.

## Overview and Op Script

The following script, which is shown in both XSLT and SLAX formats, restarts an FPC given the slot number in which the FPC resides. The user provides the slot number in the command-line interface (CLI) when the script is invoked. The script stores the slot number as the value of the parameter **slot** and constructs the **request chassis fpc** command string to include the slot number of the FPC to restart. There is no Junos Extensible Markup Language (XML) equivalent for the **request chassis** commands. Therefore, this script invokes the **request chassis fpc** command directly rather that using a remote procedure call (RPC).

**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="arguments">
    <argument>
      <name>slot</name>
      <description>Slot number of the FPC</description>
    </argument>
  </xsl:variable>
  <xsl:param name="slot"/>
  <xsl:template match="/">
    <op-script-results>
      <xsl:variable name="restart">
        <command>
          <xsl:value-of select="concat('request chassis fpc slot ', $slot,'
                                                    restart')"/>
        </command>
      </xsl:variable>
      <xsl:variable name="result" select="jcs:invoke($restart)"/>
      <output>
        <xsl:text>Restarting the FPC in slot </xsl:text>
        <xsl:value-of select="$slot"/>
        <xsl:text>. </xsl:text>
        <xsl:text>To verify, issue the "show chassis fpc" command.</xsl:text>
      </output>
    </op-script-results>
  </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 $arguments = {
  <argument> {
    <name> "slot";
```

```
    <description> "Slot number of the FPC";
  }
}
param $slot;
match / {
  <op-script-results> {
    var $restart = {
      <command> 'request chassis fpc slot ' _ $slot _ ' restart';
    }
    var $result = jcs:invoke($restart);
    <output> {
      expr "Restarting the FPC in slot ";
      expr $slot;
      expr ". ";
      expr "To verify, issue the \"show chassis fpc\" command.";
    }
  }
}
```

## 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 **restart-fpc.xsl** or **restart-fpc.slax** as appropriate, and download it to the **/var/db/scripts/op/** directory on the device.

   Only users who belong to the Junos OS **super-user** login class can access and edit files in this directory.

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

   ```
   [edit system scripts op]
   user@host# set file restart-fpc.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op restart-fpc slot *slot-number*** operational mode command.

## Verification

### Verifying Op Script Execution

**Purpose** Verify that the FPC has been restarted and is currently online.

**Action** Execute the op script by issuing the **op *filename*** operational mode command. Supply the **slot** number of the FPC as an argument.

```
user@host> op restart-fpc slot 0
```

When you execute the script, you should see output similar to the following:

```
Restarting the FPC in slot 0. To verify, issue the "show chassis fpc" command.
```

Issue the **show chassis fpc detail *fpc-slot*** operational mode command.

```
user@host> show chassis fpc detail 0

Slot 0 information:
  State                                   Online
  Temperature                          36 degrees C / 96 degrees F
  Total CPU DRAM                     1024 MB
  Total RLDRAM                        256 MB
  Total DDR DRAM                     4096 MB
  Start time:                             2009-08-11 21:20:30 PDT
  Uptime:                                 0 hours, 1 minutes, 50 seconds
  Max Power Consumption               335 Watts
```

**Meaning** The **show chassis fpc detail** command output displays the state, start time, uptime, and characteristics for the FPC. Verify that the FPC was restarted by checking the start time and uptime of the FPC. Verify the status of the restarted FPC by checking the state. If the status is **Present**, the FPC is coming up but is not yet online. If the status is **Online**, the FPC is online and running.

## Example: Searching Files Using an Op Script

This sample script searches a file on a device running Junos OS for lines matching a given regular expression. The example uses the **jcs:grep** template in an op script.



### Requirements

This example uses a device running Junos OS.

### Overview and Op Script

The **jcs:grep** template searches an ASCII file for lines matching a regular expression. The template resides in the **junos.xsl** import file, which is included with the standard Junos OS installation available on all switches, routers, and security devices running Junos OS. To use the **jcs:grep** template in a script, you must import the **junos.xsl** file into the script and map the **jcs** prefix to the namespace identified by the URI http://xml.juniper.net/junos/commit-scripts/1.0 .

In this example, all values required for the **jcs:grep** template are defined as global parameters. The values for the parameters are passed into the script as command-line arguments. The following script defines two parameters, **filename** and **pattern**, which store the values of the input file path and the regular expression. If you omit either argument when you execute the script, the script generates an error and halts execution. Otherwise, the script calls the **jcs:grep** template and passes in the supplied arguments.

If the regular expression contains a syntax error, the **jcs:grep** template generates an **error: regex error** message for each line in the file. If the regular expression syntax is valid, the template parses the input file. For each match, the template adds a **<match>** element, which contains **<input>** and **<output>** child tags, to the result tree. The template writes the matching string to the **<output>** child element and writes the corresponding matching line to the **<input>** child element:

```
<match> {
  <input>
  <output>
}
```

In the SLAX script, the **:=** operator copies the results of the **jcs:grep** template call to a temporary variable and runs the **node-set** function on that variable. The **:=** operator ensures that the **results** variable is a node-set rather than a result tree fragment so that the script can access the contents. The XSLT script explicitly calls out the equivalent steps. The script then loops through all resulting input elements and prints each match.

**XSLT Syntax**

```
<?xml version="1.0" standalone="yes"?>
<xsl:stylesheet 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" version="1.0">
  <xsl:import href="../import/junos.xsl"/>

  <xsl:variable name="arguments">
    <argument>
      <name>filename</name>
      <description>name of file in which to search for the specified pattern
      </description>
    </argument>
    <argument>
      <name>pattern</name>
      <description>regular expression</description>
    </argument>
  </xsl:variable>

  <xsl:param name="filename"/>
  <xsl:param name="pattern"/>

  <xsl:template match="/">

    <op-script-results>
      <xsl:choose>
        <xsl:when test="$filename = ''">
          <xnm:error>
            <message>missing mandatory argument 'filename'</message>
          </xnm:error>
```

```
            </xsl:when>
            <xsl:when test="$pattern = '';">
              <xnm:error>
                <message>missing mandatory argument 'pattern'</message>
              </xnm:error>
            </xsl:when>
            <xsl:otherwise>
              <xsl:variable name="results-temp">
                <xsl:call-template name="jcs:grep">
                  <xsl:with-param name="filename" select="$filename"/>
                  <xsl:with-param name="pattern" select="$pattern"/>
                </xsl:call-template>
              </xsl:variable>
              <xsl:variable xmlns:ext="http://xmlsoft.org/XSLT/namespace"
                name="results" select="ext:node-set($results-temp)"/>
              <output>
                <xsl:value-of select="concat('Search for ', $pattern, ' in ', $filename)"/>
              </output>
              <xsl:for-each select="$results//input">
                <output>
                  <xsl:value-of select="."/>
                </output>
              </xsl:for-each>
            </xsl:otherwise>
          </xsl:choose>
        </op-script-results>

    </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 $arguments = {
    <argument> {
      <name> "filename";
      <description> "name of file in which to search for the specified pattern";
    }
    <argument> {
      <name> "pattern";
      <description> "regular expression";
    }
}

param $filename;
param $pattern;

match / {
    <op-script-results> {

        if ($filename = '') {
            <xnm:error> {
```

```
            <message> "missing mandatory argument 'filename'";
          }
        }
        else if ($pattern = '') {
          <xnm:error> {
            <message> "missing mandatory argument 'pattern'";
          }
        }
        else {
          var $results := { call jcs:grep($filename, $pattern);  }

          <output> "Search for " _ $pattern _ " in " _ $filename;
          for-each ($results//input) {
            <output> .;
          }
        }
      }
    }
```

## 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 **grep.xsl** or **grep.slax** as appropriate, and download it to the **/var/db/scripts/op/** directory on the device.
2. In configuration mode, include the **file** statement at the **[edit system scripts op]** hierarchy level and **grep.xsl** or **grep.slax** as appropriate.

   ```
   [edit system scripts op]
   user@host# set file grep.(slax | xsl)
   ```

3. Issue the **commit and-quit** command to commit the configuration and to return to operational mode.

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

4. Execute the op script by issuing the **op grep filename** ***filename*** **pattern** ***pattern*** operational mode command.

## Verification

### Verifying the Script Arguments

**Purpose** Verify that the argument names and descriptions appear in the command-line interface (CLI) help.

**Action** Issue the **op grep ?** operational mode command. The CLI lists the possible completions for the script arguments based on the definitions within the global variable **arguments** in the script.

```
user@host> op grep
Possible completions:
  <[Enter]>            Execute this command
  <name>               Argument name
  detail               Display detailed output
  filename             name of file in which to search for the specified pattern

  pattern              regular expression
  |                    Pipe through a command
```

### Verifying Op Script Execution

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

**Action** If you issue the **op grep** command, but you fail to supply either the filename or the regex pattern, the script issues an error message and halts execution. For example:

```
user@host> op grep filename /var/log/messages
error: missing mandatory argument 'pattern'
```

```
user@host> op grep pattern SNMP_TRAP_LINK_DOWN
error: missing mandatory argument 'filename'
```

When you issue the **op grep filename *filename* pattern *pattern*** command, the script lists all lines from the input file that match the regular expression.

```
user@host> op grep filename /var/log/messages pattern SNMP_TRAP_LINK_DOWN
Search for SNMP_TRAP_LINK_DOWN in /var/log/messages
Feb 24 09:04:00  host mib2d[1325]: SNMP_TRAP_LINK_DOWN: ifIndex 543, ifAdminStatus
 down(2), ifOperStatus down(2), ifName
lt-0/1/0.9
Feb 24 09:04:00  host mib2d[1325]: SNMP_TRAP_LINK_DOWN: ifIndex 542, ifAdminStatus
 down(2), ifOperStatus down(2), ifName
lt-0/1/0.10
```

**Related Documentation**

- SLAX Templates Overview on page 44
- jcs:grep Template on page 131
- regex() Function (jcs and slax Namespaces) on page 118

CHAPTER 27

# Summary of Op Script Configuration Statements

This chapter describes each configuration statement for operation (op) scripts. The statements are organized alphabetically.

## arguments (Op Scripts)

**Syntax**

```
arguments {
  argument-name {
    description descriptive-text;
  }
}
```

**Hierarchy Level** [edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.

**Description** For Junos OS op scripts, configure command-line arguments to the script.

**Options** ***argument-name***—The name of a command-line argument to an op script.

The remaining statement is explained separately.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Declaring Arguments in Op Scripts on page 469
- description on page 539

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 282
- Configuring Checksum Hashes for an Event Script on page 677
- Configuring Checksum Hashes for an Op Script on page 475
- Executing an Op Script from a Remote Site on page 476
- 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*

## command

**Syntax** command *filename-alias*;

**Hierarchy Level** [edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, configure a filename alias for the script file. This allows you to run the script by referencing either the script filename or the filename alias.

**Options** ***filename-alias***—Alias for the script file.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Enabling an Op Script and Defining a Script Alias on page 474

## description

**Syntax** description *descriptive-text*;

**Hierarchy Level** [edit system scripts op file *filename*]
[edit system scripts op file *filename* arguments *argument-name*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, provide a help-text string that appears in the command-line interface (CLI).

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Configuring Help Text for Op Scripts on page 471
- Declaring Arguments in Op Scripts on page 469
- file (Op Scripts) on page 540

## file (Op Scripts)

**Syntax**

```
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;
}
```

**Hierarchy Level** [edit system scripts op]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, enable an op script that is located in the **/var/db/scripts/op** directory.

**Options** *filename*—The name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an op script.

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**

- Enabling an Op Script and Defining a Script Alias on page 474

## max-datasize

**Syntax** max-datasize *size*;

**Hierarchy Level** [edit event-options event-script],
[edit system scripts commit],
[edit system scripts op]

**Release Information** Statement introduced in Junos OS Release 12.3.

**Description** Maximum amount of memory allocated for the data segment during execution of a script of the given type. Junos OS sets the maximum memory limit for the executing script to the configured value irrespective of the total memory available on the system at the time of execution. If the executing script exceeds the specified maximum memory limit for that script type, it exits gracefully.

**Default** If you do not include the **max-datasize** statement, the system allocates half of the total available memory of the system up to a maximum value of 128 MB for the data segment portion of the executed script.

**Options** ***size***—Maximum amount of memory allocated for the data segment during execution of a script of the given type. If you do not specify a unit of measure, the default is bytes.

**Syntax:** ***size*** to specify bytes, ***size*****k** to specify KB, ***size*****m** to specify MB, or ***size*****g** to specify GB

**Range:** 2,3068,672 (22 MB) through 1,073,741,824 (1 GB)

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- max-policies on page 640
- Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts on page 227

## no-allow-url

**Syntax** no-allow-url;

**Hierarchy Level** [edit system scripts op]

**Release Information** Statement introduced in Junos OS Release 10.0.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, prohibit the remote execution of scripts. When you include this configuration statement, the **op url** operational mode command generates an error and does not permit you to execute the op script from a remote site.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- file (Op Scripts) on page 540
- Executing an Op Script from a Remote Site on page 476

## op

**Syntax**

```
op {
  file filename {
    arguments {
      argument-name {
        description descriptive-text;
      }
    }
    checksum (md5 | sha-256 | sha1) hash;
    command filename-alias;
    description descriptive-text;
    max-datasize
    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 scripts]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, configure an operation 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 on page 213

## refresh (Op Scripts)

**Syntax** refresh;

**Hierarchy Level** [edit system scripts op],
[edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 on the QFX Series.

**Description** For Junos OS op scripts, overwrite the local copy of all enabled op scripts or a single enabled script located in the **/var/db/scripts/op** directory with the copy located at the source URL, specified in the **source** statement at the same hierarchy level.

NOTE: On the QFabric system, op scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/op/** directory on the Director device.

**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 on page 218
- refresh-from (Op Scripts) on page 545
- source (Op Scripts) on page 546

## refresh-from (Op Scripts)

**Syntax** refresh-from *url*;

**Hierarchy Level** [edit system scripts op],
[edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 on the QFX Series.

**Description** For Junos OS op scripts, overwrite the local copy of all enabled op scripts or a single enabled script located in the **/var/db/scripts/op** directory with the copy located at a URL other than the URL specified in the **source** statement.

NOTE: This statement is not supported on the QFabric system.

**Options** *url*—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 on page 222
- refresh (Op Scripts) on page 544
- source (Op Scripts) on page 546

## scripts

**See** scripts

## source (Op Scripts)

**Syntax** source *url*;

**Hierarchy Level** [edit system scripts op file *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**Description** For Junos OS op scripts, specify the location of the source file for an enabled script located in the **/var/db/scripts/op** directory. When you include the **refresh** statement at the same hierarchy level, the local copy is overwritten by the version stored at the specified URL.

NOTE: On the QFabric system, commit scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/op/** directory on the Director device.

**Options** ***url***—Master source file for an op script 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 on page 218
- refresh (Op Scripts) on page 544
- refresh-from (Op Scripts) on page 545

## traceoptions

**See** traceoptions (Commit and Op Scripts)

PART 5

# Event Policy

CHAPTER 28

# Event Policy Overview

This chapter discusses the following topics:



## Event Notifications and Policies Overview

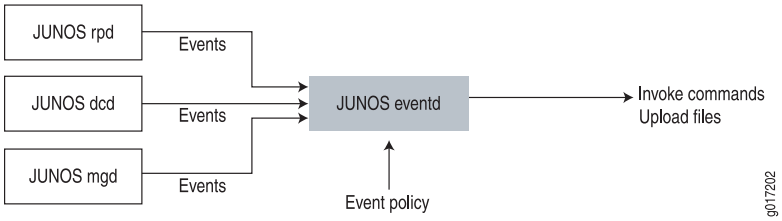
To diagnose a fault or error condition on a device, you need relevant information about the state of the platform. You can derive state information from *event notifications*. Event notifications are system log messages and SNMP traps. A Junos OS process called the *event process* (eventd) receives event notifications—henceforth simply called *events*—from other Junos OS processes.

Timely diagnosis and intervention can correct error conditions and keep the device in operation. After the eventd process receives events, *event policies* instruct the eventd process to select specific events, correlate the events, and perform a set of actions. These actions can either help you diagnose a fault or take corrective action. For example, the eventd process can upload device files to a given destination and issue operational mode commands.

Events can originate as SNMP traps or system log messages. The event process receives event messages from other Junos OS processes, such as the routing protocol process (rpd) and the management process (mgd). Depending on the custom event policy you configure, eventd listens for specific events and in response to these events might create a log file, invoke a Junos command, or invoke an event script. When an event script is invoked, event details are passed to the event script in the form of XML inputs.

Figure 11 on page 549 shows how the event process (eventd) interacts with other Junos OS processes.

### Figure 11: Interaction of eventd Process with Other Junos OS Processes



## How Event Policies Work

An event policy is an if-then-else construct. It defines actions to be executed by the eventd process on receipt of an event. You can configure multiple policies to be processed for an event. The policies are executed in the order in which they appear in the configuration. For each policy, you can configure multiple actions. The actions are also executed in the order in which they appear in the configuration.

To view a list of the events that can be referenced in an event policy, issue the **help syslog ?** command:

```
user@host> help syslog ?
Possible completions:
  <syslog-tag>         System log tag
  ACCT_ACCOUNTING_FERROR  Error occurred during file processing
  ACCT_ACCOUNTING_FOPEN_ERROR  Open operation failed on file
  ...
```

You can filter the output of a search by using the pipe (**|**) symbol. The following example lists the filters that can be used with the pipe symbol:

```
user@host> help syslog | ?
Possible completions:
  count                Count occurrences
  display              Show additional kinds of information
  except               Show only text that does not match a pattern
  find                 Search for first occurrence of pattern
  hold                 Hold text without exiting the --More-- prompt
  last                 Display end of output only
  match                Show only text that matches a pattern
  no-more              Don't paginate output
  request              Make system-level requests
  resolve              Resolve IP addresses
  save                 Save output text to file
  trim                 Trim specified number of columns from start of line
```

For more information about using the pipe symbol, see the CLI User Guide.

You can also list multiple events as you configure the event policy. To view a partial list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
 <event>
 [            Open a set of values
 acct_accounting_ferror
 acct_accounting_fopen_error
 ...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see "Using Nonstandard System Log Messages to Trigger Event Policies" on page 561.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events to Trigger Event Policies" on page 560.

In response to events, the eventd process can correlate two or more events based on a policy, and execute the following actions:

- Ignore the event—Do not generate a system log message for this event and do not process any further policy instructions for this event.
- Upload a file—Upload a file to a specified destination. You can specify a transfer delay, so that, on receipt of an event, the upload of the file begins after the configured transfer delay. For example, to upload a core file, a transfer delay can ensure that the core file has been completely generated before the upload begins.
- Execute Junos OS operational mode commands—Execute commands on receipt of an event. The XML or text output of these commands is stored in a file, which is then uploaded to a specified URL. You can include variables in the command that allow data from the triggering event to be automatically included in the command syntax.
- Execute Junos OS configuration mode commands—Execute commands to modify the configuration on receipt of an event. Starting with Junos OS Release 12.1, you can configure an event policy to modify the configuration using Junos OS configuration mode commands and then commit the updated configuration.
- Execute Junos OS event scripts—Execute event scripts on receipt of an event. Event scripts are Extensible Stylesheet Transformation (XSLT) or Stylesheet Language Alternative Syntax (SLAX) scripts that you write to perform any function available through Junos XML or Junos XML protocol remote procedure calls (RPCs). Additionally, you can pass to an event script a set of arguments that you define. A script can build and run an operational mode command, receive the command output, inspect the output, and determine the next appropriate action. This process can be repeated until the source of the problem is determined. The output of the scripts is stored in a file, which is then uploaded to a specified URL. You can include variables in the arguments to the scripts that allow data from the triggering event to be incorporated into the script.
- Raise an SNMP trap.

CHAPTER 29

# Configuring Event Policy

Event policies can monitor specific events, create log files, invoke Junos OS commands, and invoke event scripts. This chapter discusses the command-line interface (CLI) statements for configuring event policies.

To configure event policy, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
destinations {
    destination-name {
        archive-sites {
            url <password password>;
        }
        transfer-delay seconds;
    }
}
generate-event event-name {
    time-interval seconds;
    time-of-day hh:mm:ss;
}
policy policy-name {
    attributes-match {
        event1.attribute-name equals event2.attribute-name;
        event.attribute-name matches regular-expression;
        event1.attribute-name starts-with event2.attribute-name;
    }
    events [ events ];
    then {
        change-configuration {
            commands {
                "command";
            }
            commit-options {
                check <synchronize>;
                force;
                log "comment-string";
                synchronize;
            }
            retry count number interval seconds;
            user-name username;
        }
        event-script filename {
```

```
        arguments {
          argument-name argument-value;
        }
        destination (Event Policy) destination-name {
          retry-count number retry-interval seconds;
          transfer-delay seconds;
        }
        output-filename filename;
        output-format (text | xml);
        user-name name;
      }
      execute-commands {
        commands {
          "command";
        }
        destination destination-name {
          retry-count number retry-interval seconds;
          transfer-delay seconds;
        }
        output-filename filename;
        output-format (text | xml);
        user-name username;
      }
      ignore;
      priority-override {
        facility facility-type;
        severity severity-type;
      }
      raise-trap;
      upload filename (filename | committed) destination destination-name {
        retry-count number retry-interval seconds;
        transfer-delay seconds;
        user-name username;
      }
    }
    within seconds {
      events [ events ];
      not events [ events ];
      trigger (on | after | until) event-count;
    }
  }
  traceoptions {
    file <filename> <files number> <match regular-expression> <size size> <world-readable |
      no-world-readable>;
    flag flag;
    no-remote-trace;
  }
```

This chapter discusses the following topics:

## Using Correlated Events to Trigger an Event Policy

You can configure a policy that correlates two or more events. If the correlated events occur as specified, they cause particular actions to be taken. For example, you might want to issue certain operational mode commands when a **UI_CONFIGURATION_ERROR** event is generated within five minutes (300 seconds) after a **UI_COMMIT_PROGRESS** event. As another example, you might want to upload a particular file if a **DCD_INTERFACE_DOWN** event is generated two times within a 60-second interval.

To configure a policy that correlates events, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
policy policy-name {
  events [ events ];
  within seconds {
    events [ events ];
    not events [ events ];
    trigger (on | after | until) event-count;
  }
  attributes-match {
    event1.attribute-name equals event2.attribute-name;
    event.attribute-name matches regular-expression;
    event1.attribute-name starts-with event2.attribute-name;
  }
  then {
    ...
  }
}
```

In the **events** statement, you can list multiple events. To view a list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
user@host# set event-options policy policy-name events ?
Possible completions:
 <event>
 [              Open a set of values
 acct_accounting_ferror
 acct_accounting_fopen_error
 ...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see "Using Nonstandard System Log Messages to Trigger Event Policies" on page 561.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events to Trigger Event Policies" on page 560.

The actions configured in the **then** statement are executed only if certain conditions are met, which you specify in the **within** and **attributes-match** statements.

You can configure a policy that is executed only if a specified event occurs within a specified time interval after another event. You do this by including the **within *seconds* events** statement. The policy is executed only if one or more of the events in the first **events** statement occur within a configured number of seconds after one or more of the events in the **within *seconds* events** statement. The number of seconds can be from 60 through 604,800. The **not** statement causes the policy to be executed only if the events do not occur within the configured time interval.

For example, the following policy is executed if ***event3***, ***event4***, or ***event5*** occurs within 60 seconds after ***event1*** or ***event2*** occurs:

```
[edit event-options]
policy 1 {
   events [ event3 event4 event5 ];
   within 60 events [ event1 event2 ];
   then {
      ...
   }
}
```

The **attributes-match** statement correlates two events as follows:

- ***event1.attribute-name* equals *event2.attribute-name***—Execute the policy only if the specified attribute of ***event1*** equals the specified attribute of ***event2***.
- ***event.attribute-name* matches *regular-expression***—Execute the policy only if the specified attribute of ***event*** matches a regular expression. For more information, see "Using Regular Expressions to Refine the Set of Events That Trigger a Policy" on page 559.
- ***event1.attribute-name* starts-with *event2.attribute-name***—Execute the policy only if the specified attribute of ***event1*** starts with the specified attribute of ***event2***.

If the **attributes-match** statement includes the **equals** or **starts-with** options, or if it includes a **matches** option that includes a clause for an event that is not specified at the **[edit event-options policy *policy-name* events]** hierarchy level, you must include one or more **within** statements in the same policy configuration.

Starting with Junos OS Release 11.1, you can use event policy variables within the **attributes-match** statement to differentiate between a trigger event attribute and a correlated event attribute. The double dollar sign (**$$**) notation represents the event that is triggering a policy, and **{$$.*attribute-name*}** resolves to the value of the attribute of the triggering event. Triggering events are those that you configure at the **[edit event-options policy *policy-name* events]** hierarchy level. For correlating events, the single dollar sign with the event name (**$*event***) notation represents the most recent event that matches the event name, and **{$*event.attribute-name*}** resolves to the value of the attribute of the correlated event.

In the following example, the policy will execute the actions under the **then** statement if four or more commits are performed within a 5-minute period, and the username of one or more of the correlated events is the same as the username of the trigger event.

```
policy multiple-commits {
  events ui_commit;
  attributes-match {
    {$$.user-name} equals {$ui_commit.user-name};
  }
  within 300 {
    trigger after 3;
    events ui_commit;
  }
  then ...
}
```

To view a list of all event attributes that you can reference, issue the **help syslog *event*** operational mode command. The output of this command shows the event attributes in angle brackets (< >). The following output shows that three attributes can be referenced for the **ACCT_ACCOUNTING_SMALL_FILE_SIZE** event: **filename**, **file-size**, and **record-size**.

```
user@host> help syslog ACCT_ACCOUNTING_SMALL_FILE_SIZE
Name:           ACCT_ACCOUNTING_SMALL_FILE_SIZE
Message:       File <filename> size (<file-size>) is smaller than record size
(<record-size>)
```

You can filter the output of a search by using the pipe (**|**) symbol. The following example lists the filters that can be used with the pipe symbol:

```
user@host> help syslog | ?
Possible completions:
  count                Count occurrences
  display              Show additional kinds of information
  except               Show only text that does not match a pattern
  find                 Search for first occurrence of pattern
  hold                 Hold text without exiting the --More-- prompt
  last                 Display end of output only
  match                Show only text that matches a pattern
  no-more              Don't paginate output
  request              Make system-level requests
  resolve              Resolve IP addresses
```

```
save                Save output text to file
trim                Trim specified number of columns from start of line
```

For more information about using the pipe symbol, see the CLI User Guide.

Another way to view the attributes you can reference is by issuing the **set attributes-match *event*?** command at the **[edit event-options policy *policy-name*]** hierarchy level, as shown in the following example:

```
[edit event-options policy p1]
user@host# set attributes-match acct_accounting_small_file_size?
Possible completions:
 <from-event-attribute>  First attribute to compare
 acct_accounting_small_file_size.filename
 acct_accounting_small_file_size.filesize
 acct_accounting_small_file_size.record-size
```

NOTE: In this set command, there is no space between the event name and the question mark (?).

**Related Documentation**

- Representing the Correlating Event in an Event Policy on page 558
- Triggering an Event Policy Based on Event Count on page 559
- Using Regular Expressions to Refine the Set of Events That Trigger a Policy on page 559
- attributes-match on page 626
- policy (Event Policy) on page 643
- not on page 641
- then on page 650
- within on page 659

## Representing the Correlating Event in an Event Policy

As described in "Configuring an Event Policy to Execute Operational Mode Commands" on page 574, the double dollar sign ($$) notation represents the event that is triggering a policy. Triggering events are those that you configure at the **[edit event-options policy *policy-name* events]** hierarchy level.

As described in "Using Correlated Events to Trigger an Event Policy" on page 555, you can configure a policy that is executed only if a specified event occurs within a specified time interval after another event. You do this by including the **within *seconds* events** statement at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name ]
events [ events ];
within seconds events [ events ];
```

The policy is executed only if one or more of the events at the **[edit event-options policy *policy-name* events]** hierarchy level occur within a configured number of seconds after one or more of the events in the **within *seconds* events** statement.

For correlating events, the single dollar sign with the event name (**$*event***) notation represents the most recent event that matches the event name. The dollar sign with the asterisk (**$***) notation represents the most recent event that matches any of the correlating events.

For a configuration example, see "Example: Representing the Correlating Event in an Event Policy" on page 617.

## Triggering an Event Policy Based on Event Count

You can configure an event policy to be triggered if an event or set of events occurs a specified number of times within a specified time period.

To do this, include the optional **trigger** statement at the **[edit event-options policy *policy-name* within *seconds*]** hierarchy level:

```
[edit event-options policy policy-name within seconds]
trigger (after | on | until) event-count;
```

The software counts the number of times the triggering event occurs. A triggering event can be any event configured at the **[edit event-options policy *policy-name* events]** hierarchy level. You can configure the following options:

- **after *event-count***—The policy is executed when the number of matching events received equals ***event-count*** plus one.
- **on *event-count***—The policy is executed when the number of matching events received equals ***event-count***.
- **until *event-count***—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals ***event-count***.

For a configuration example, see "Example: Triggering a Policy Based on Event Count" on page 619.

## Using Regular Expressions to Refine the Set of Events That Trigger a Policy

You can use regular expression matching to specify more exactly which events cause a policy to be executed.

To specify the text string that must appear in an event attribute for the policy to be executed, include the **matches** statement at the **[edit event-options policy *policy-name* attributes-match]** hierarchy level, and specify the regular expression which the event attribute must match:

```
[edit event-options policy policy-name attributes-match]
event.attribute-name matches regular-expression;
```

When you specify the regular expression, use the notation defined in POSIX Standard 1003.2 for extended (modern) UNIX regular expressions. Explaining regular expression syntax is beyond the scope of this document. Table 38 on page 560 specifies which character or characters are matched by some of the regular expression operators that you can use in the **matches** statement. In the descriptions, the term *term* refers to

either a single alphanumeric character or a set of characters enclosed in square brackets, parentheses, or braces.

NOTE: The matches statement is not case-sensitive.

Table 38: Regular Expression Operators for the matches Statement

| Operator | Matches |
|---|---|
| . (period) | One instance of any character except the space. |
| * (asterisk) | Zero or more instances of the immediately preceding term. |
| + (plus sign) | One or more instances of the immediately preceding term. |
| ? (question mark) | Zero or one instance of the immediately preceding term. |
| \| (pipe) | One of the terms that appear on either side of the pipe operator. |
| ! (exclamation point) | Any string except the one specified by the expression, when the exclamation point appears at the start of the expression. Use of the exclamation point is specific to Junos OS. |
| ^ (caret) | The start of a line, when the caret appears outside square brackets.<br><br>One instance of any character that does not follow it within square brackets, when the caret is the first character inside square brackets. |
| $ (dollar sign) | The end of a line. |
| **[ ]** (paired square brackets) | One instance of one of the enclosed alphanumeric characters. To indicate a range of characters, use a hyphen ( - ) to separate the beginning and ending characters of the range. For example, **[a-z0-9]** matches any letter or number. |
| **( )** (paired parentheses) | One instance of the evaluated value of the enclosed term. Parentheses are used to indicate the order of evaluation in the regular expression. |

For a configuration example, see "Example: Controlling Event Policy Using a Regular Expression" on page 614.

## Generating Internal Events to Trigger Event Policies

*Internal events* are events that you create to trigger a policy to be executed. They are not generated by Junos OS processes, and they do not have any associated system log messages. You can generate an internal event based on a time interval or the time of day.

To generate an event, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
generate-event event-name {
  time-interval seconds;
  time-of-day hh:mm:ss;
}
```

In the **time-interval** statement, configure a frequency, in seconds, with which to repeatedly generate an event. The time interval can range from 60 through 2,592,000 seconds.

In the **time-of-day** statement, configure a time of day for the event to occur. Use the format ***hh*:*mm*:*ss***.

NOTE: If you modify the system time by issuing the **set date** operational mode command, we recommend that you also issue the **commit full** or the **restart event-process** command. Otherwise, an internal event based on the time of day might not be generated at the configured time.

For example, if you configure an internal event to be generated at 15:55:00, and then you modify the system time from 15:47:17 to 15:53:00, the event is generated when the system time is approximately 16:00 instead of at the configured time, 15:55:00. You can correct this problem by issuing the **commit full** or the **restart event-process** command.

You can configure up to 10 internal events. If you attempt to commit a configuration with more than 10 internal events, Junos OS generates an error, and the commit fails.

For configuration examples, see "Example: Generating an Internal Event" on page 616.

## Using Nonstandard System Log Messages to Trigger Event Policies

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. These system log messages have an event ID and a **message** attribute. Event IDs are based on the origin of the message, as shown in Table 39 on page 561.

### Table 39: Event ID by System Log Message Origin

| Event IDs | Origin |
|---|---|
| SYSTEM | Messages from Junos daemons and utilities |
| KERNEL | Messages from the Junos kernel |
| PIC | Messages from physical interface cards (PICs) |
| PFE | Messages from the Packet Forwarding Engine |
| LCC | On a TX Matrix router, messages from a line-card chassis (LCC) |

Table 39: Event ID by System Log Message Origin *(continued)*

| Event IDs | Origin |
|---|---|
| SCC | On a TX Matrix router, messages from a switch-card chassis (SCC) |

To base your event policy on the event types shown in Table 39 on page 561, include the **events *event-id*** statement and the **attributes-match** statement with the ***event-id*.message matches "*message*"** attribute at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
events event-id;
attributes-match {
  event-id.message matches "message";
}
```

For a configuration example, see "Example: Using Nonstandard System Log Messages to Trigger an Event Policy" on page 621.

## Event Policy File Archiving Overview

Various types of files are useful in diagnosing events. When an event policy action generates output files, you can archive the files for later analysis. Similarly, you might want to archive system files, including system log files, core files, and configuration files, from the time an event occurs.

When an event occurs, you can upload relevant files to a specified location for analysis. To archive files from an event policy, configure one or more *destinations* specifying the archive sites to which the files are uploaded. You then reference the configured destinations within event policies.

A transfer delay allows you to specify the number of seconds the event process (eventd) waits before beginning to upload a file or multiple files. A transfer delay helps ensure that a large file, such as a core file, is completely generated before the upload begins.

You can associate transfer delays with a destination and with an event policy action. If you associate a transfer delay with a destination, the transfer delay applies to all file upload actions that use that destination. You can also assign a transfer delay to a single event policy action. For example, you might have multiple event policy actions that use the same destination, and for some of these event policy actions, you want a transfer delay, and for other event policy actions you want no transfer delay. If you configure a transfer delay for a destination, and you also configure a transfer delay for the event policy action, the resulting transfer delay is the sum of the two delays.

Transient network problems can cause a file upload operation to fail. By default, if the file upload operation fails for any reason, the event policy does not retry the upload operation. However, you can configure an event policy to retry the file upload operation a specified number of times if the initial upload fails. You can also configure the time interval between each retry attempt.

**Related Documentation**

- Example: Defining Destinations for File Archiving by Event Policies on page 563
- Example: Configuring an Event Policy to Upload Files on page 566
- Configuring the Delay Before Files Are Uploaded by an Event Policy on page 572
- Configuring an Event Policy to Retry the File Upload Action on page 574

## Example: Defining Destinations for File Archiving by Event Policies

This example configures an archive site for event policies. Event policy actions that reference the configured destination upload specified files to that site.



### Requirements

This example uses a device running Junos OS. No additional configuration beyond device initialization is required before configuring this example.

### Overview

When an event policy action generates output files, you can archive the files for later analysis. Similarly, you might want to archive system files, including system log files, core files, and configuration files, from the time an event occurs.

When an event occurs, you can upload relevant files to a specified location for analysis. To archive files from an event policy, configure one or more *destinations* specifying the archive sites to which the files are uploaded. You then reference the configured destinations within event policies.

To define a destination archive site, include the **destinations** statement at the **[edit event-options]** hierarchy level.

```
[edit event-options]
destinations {
  destination-name {
    archive-sites {
      url <password password>;
    }
    transfer-delay seconds;
  }
}
```

The *destination-name* is a user-defined identifier, which is referenced by event policies. You can define multiple destinations with different archive sites.

For each destination, configure one or more archive site URIs, which are the actual sites to which the files are uploaded. If you specify multiple archive site URIs for a given destination, the device attempts to transfer to the first archive site in the list, moving to

the next site in the list only if the transfer fails. Optionally, you can specify a plain-text password for login into an archive site.

Specify the archive site URI as a file URI, an active or passive FTP URI, or a Secure Copy (SCP) URI. Local device directories are also supported (for example, **/var/tmp**). When you specify the archive site URI, do not add a forward slash (/) to the end of the URI.

file:<//host>/path

ftp://username@host:<port>url-path

pasvftp://username@host:<port>url-path

scp://username@host:<port>url-path

<path>/<filename>

The format for the destination filename is ***device-name_filename_YYYYMMDD_HHMMSS***.

The **transfer-delay** statement allows you to specify the number of seconds the event process (eventd) waits before beginning to upload a file or multiple files to that destination. A transfer delay allows you to ensure that a large file, such as a core file, is completely generated before the upload begins. For more information, see "Configuring the Delay Before Files Are Uploaded by an Event Policy" on page 572.

This example configures a new archive destination named mgmt-archives, which can be referenced in event policies for file archiving. The example configures two archive sites for this destination. The first site is the Secure Copy URI "scp://username@example.com/test" for which a password is configured. The second site is a directory on the local device. The device attempts to transfer to the first archive site in the list, moving to the next site only if the transfer to the first site fails. The example configures a transfer delay of five seconds for all files uploaded to the mgmt-archives archive site.

## Configuration

**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set event-options destinations mgmt-archives archive-sites
   "scp://username@example.com/test" password PaSsWoRd
set event-options destinations mgmt-archives archive-sites /var/log
set event-options destinations mgmt-archives transfer-delay 5
```

**Step-by-Step Procedure**

Configure a new archive destination named mgmt-archives that can be referenced by event-policies.

1. Configure the identifier and associated archive sites for each destination.

   The device transfers to the first archive site in the list, moving to the next site only if the transfer to the first site fails.

   ```
   [edit event-options destinations]
   user@host# set mgmt-archives archive-sites scp://username@example.com/test
   ```

```
user@host# set mgmt-archives archive-sites /var/log
```

2. If authentication is required to access any of the archive sites, configure the required plain-text password for that site.

```
[edit event-options destinations]
user@host# set mgmt-archives archive-sites scp://username@example.com/test
  password PaSsWoRd
```

3. (Optional) Configure the transfer delay associated with each destination. The mgmt-archives destination has a transfer delay of five seconds.

```
[edit event-options destinations]
user@host# set mgmt-archives transfer-delay 5
```

4. Commit the configuration.

```
user@host# commit
```

5. You can reference configured destinations in an event policy. For information about referencing destinations in event policies, see "Example: Configuring an Event Policy to Upload Files" on page 566 and "Configuring an Event Policy to Execute Operational Mode Commands" on page 574.

## Verification

### Verifying the Configuration

**Purpose** Issue the **show configuration event-options** operational mode command to review the resulting configuration.

**Action**

```
user@host> show configuration event-options
destinations {
  mgmt-archives {
    transfer-delay 5;
    archive-sites {
      "scp://username@example.com/test" password
        "$9$z3GRF9tu0IcrKO1bYoGq.0O1IEy"; ## SECRET-DATA
      /var/log;
    }
  }
}
```

**Meaning** In the sample output, the mgmt-archives destination has two archive sites and a transfer delay of five seconds. You can now reference this destination in event policies. When you reference the mgmt-archives destination in an event policy, specified files are uploaded to the first archive site after a five second delay. If the transfer to the first archive fails, the device attempts to upload the files to the **/var/log** archive site. For more information about referencing destinations in event policies, see "Example: Configuring an Event Policy to Upload Files" on page 566.

Note that although the plain-text password is visible when you configure it, the configuration displays the encrypted password.

**Related Documentation**

- Example: Configuring an Event Policy to Upload Files on page 566
- Configuring the Delay Before Files Are Uploaded by an Event Policy on page 572
- Configuring an Event Policy to Retry the File Upload Action on page 574
- **destinations on page 632**

## Example: Configuring an Event Policy to Upload Files

This example configures event policy actions that upload relevant files to a specified location for analysis.



### Requirements

Before you begin:

- Configure the destinations that you will reference in the event policy. See "Example: Defining Destinations for File Archiving by Event Policies" on page 563.
- Configure the general event policy and triggering events.

### Overview

When an event policy action generates output files, you can archive the files for later analysis. Similarly, you might want to archive system files, including system log files, core files, and configuration files, from the time an event occurs. You can configure an event policy to upload existing system files or to upload the output files generated from an invoked event-script or command at the time an event occurs. This section outlines the configuration hierarchies for uploading each of these file types using an event policy.

When you configure an event policy to upload files, you reference configured destinations within the event policy. Specify a destination name that is configured at the **[edit event-options destinations]** hierarchy level. For more information, see "Example: Defining Destinations for File Archiving by Event Policies" on page 563.

To upload system files to a configured archive site, configure the **upload** statement at the **[edit event-options policy *policy-name* then]** hierarchy level. If the configured events occur, the eventd process executes the upload action.

```
[edit event-options policy policy-name then]
upload filename (filename | committed) destination destination-name {
  retry-count number retry-interval seconds;
  transfer-delay seconds;
  user-name username;
}
```

The **upload filename committed destination** *destination-name* statement uploads the committed configuration file.

If desired, you can include multiple **upload** statements, one for each type of file to be archived. In the **filename** statement, specify a file or multiple files to be uploaded. You can specify multiple files with one **filename** configuration statement (sometimes called *filename globbing*). For example, to upload all files that are located in the **/var/log** directory and that start with the **messages** string, include the following statement:

```
upload filename /var/log/messages* destination destination-name;
```

When an event policy executes commands in response to an event, you can write the command output to a file. To configure an event policy to upload the generated output file to a configured archive site, include the following statements at the **[edit event-options policy** *policy-name* **then]** hierarchy level:

```
[edit event-options policy policy-name then]
execute-commands {
    destination destination-name {
        retry-count count retry-interval seconds;
        transfer-delay seconds;
    }
    output-filename filename;
}
```

When an event policy executes an event script in response to an event, you can write the script output to a file. To configure an event policy to upload the generated output file to a configured archive site, include the following statements at the **[edit event-options policy** *policy-name* **then]** hierarchy level:

```
[edit event-options policy policy-name then]
event-script filename {
    destination destination-name {
        retry-count count retry-interval seconds;
        transfer-delay seconds;
    }
    output-filename filename;
}
```

The **transfer-delay** statement listed in each hierarchy defines the time interval that the system waits before uploading the files specified by that event policy action. If you have also configured a transfer delay for the destination at the **[edit event-options destinations** *destination-name***]** hierarchy level, the total transfer delay is the sum of the two delays. For more detailed information about transfer delays, see "Configuring the Delay Before Files Are Uploaded by an Event Policy" on page 572.

If the first upload attempt fails, **retry-count** specifies the number of additional times the system attempts to upload the file. The **retry-interval** specifies the time interval that the system waits between upload attempts. For more information, see "Configuring an Event Policy to Retry the File Upload Action" on page 574.

When an event policy uploads files, the files are named and time-stamped in the following format to ensure unique filenames:

*device-name_filename_YYYYMMDD_HHMMSS*

If a policy uploads multiple files within a 1-second period, the software gives each file a unique number as well, as follows:

*device-name_filename_YYYYMMDD_HHMMSS_number*

The number can be from 001 through 999. For example, if you have an event policy action with output filename **rpd-messages** on **device1**, and this event policy is executed three times in 1 second, the files are named as follows:

- **device1_rpd-messages_20070623_132333**
- **device1_rpd-messages_20070623_132333_001**
- **device1_rpd-messages_20070623_132333_002**

In this example, **policy1** consists of the following statements, where *e1* is the triggering event. The example then configures the event policy to upload a log file and the committed configuration file as well as the output files generated from the **execute-commands** and **event-script** actions.

```
[edit event-options policy policy1]
events e1;
then {
  execute-commands {
    commands {
      "show interfaces brief ge-*";
    }
  }
  event-script event-script1;
}
```

## Configuration



**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set event-options policy policy1 then upload filename /var/log/messages destination
    mgmt-archives transfer-delay 4
set event-options policy policy1 then upload filename /var/log/messages destination
    mgmt-archives retry-count 5 retry-interval 4
set event-options policy policy1 then upload filename /var/log/messages destination
    mgmt-archives user-name admin
set event-options policy policy1 then upload filename /var/log/messages destination
    mgmt-server
set event-options policy policy1 then upload filename committed destination
    mgmt-archives
set event-options policy policy1 then execute-commands output-filename ge-interfaces
set event-options policy policy1 then execute-commands destination mgmt-archives
    transfer-delay 5
```

```
set event-options policy policy1 then execute-commands destination mgmt-archives
  retry-count 5 retry-interval 4
set event-options policy policy1 then event-script event-script1 output-filename
  policy1-script-output
set event-options policy policy1 then event-script event-script1 destination mgmt-archives
  transfer-delay 5
set event-options policy policy1 then event-script event-script1 destination mgmt-archives
  retry-count 5 retry-interval 4
```

## Uploading System Files

**Step-by-Step Procedure**

Configure the event policy **policy1** to upload the system file **/var/log/messages** to the archive sites mgmt-archives and mgmt-server. Additionally, upload the committed configuration to the archive site mgmt-archives. The destination archive sites should already be configured at the **[edit event-options destinations]** hierarchy level

1. Configure the **upload** statement, and include the file to archive and the destination archive site.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set upload filename /var/log/messages destination mgmt-archives
   bsmith@R1# set upload filename /var/log/messages destination mgmt-server
   ```

2. To upload the committed configuration file, specify the filename value as **committed**.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set upload filename committed destination mgmt-archives
   ```

3. (Optional) Configure the transfer delay associated with each file and destination.

   The following configuration mode command sets the transfer delay for the **/var/log/messages** file to 4 seconds when uploaded to the mgmt-archives destination. If you have also configured a transfer delay for the destination, the total delay is the sum of the two delays.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set upload filename /var/log/messages destination mgmt-archives
     transfer-delay 4
   ```

4. (Optional) Configure the retry count and retry interval associated with a file and destination.

   In this example, if the **/var/log/messages** file fails to upload to the mgmt-archives site, the system attempts the upload up to 5 more times and waits 4 seconds in between each attempt.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1#set upload filename /var/log/messages destination mgmt-archives
     retry-count 5 retry-interval 4
   ```

5. (Optional) Configure the username associated with a file and destination. The system uploads the file using the privileges of the specified user.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1#set upload filename /var/log/messages destination mgmt-archives
     user-name admin
   ```

6. Commit the configuration.

```
[edit event-options policy policy1 then]
bsmith@R1# commit
```

### Uploading Command Output Files

**Step-by-Step Procedure**

When the event policy invokes the **execute-commands** action, the command output can be written to a file. Configure the event policy **policy1** to write command output to a file and upload the generated file to the destination mgmt-archives, which is already configured at the **[edit event-options destinations]** hierarchy level.

1. Configure the filename of the generated output file.

```
[edit event-options policy policy1 then]
bsmith@R1# set execute-commands output-filename ge-interfaces
```

2. Configure the **destination** statement to upload the generated file to the desired archive site.

```
[edit event-options policy policy1 then]
bsmith@R1# set execute-commands destination mgmt-archives
```

3. (Optional) Configure the transfer delay for each destination.

   The following command sets the transfer delay for files uploaded to the mgmt-archives destination to 5 seconds.

```
[edit event-options policy policy1 then]
bsmith@R1# set execute-commands destination mgmt-archives transfer-delay 5
```

4. (Optional) Configure the retry count and retry interval associated with each destination.

   In this example, if the output file fails to upload to the mgmt-archives site, the system attempts the upload up to 5 more times and waits 4 seconds in between each attempt.

```
[edit event-options policy policy1 then]
bsmith@R1#set execute-commands destination mgmt-archives retry-count 5
  retry-interval 4
```

5. Commit the configuration.

```
[edit event-options policy policy1 then]
bsmith@R1# commit
```

## Uploading Event Script Output Files

**Step-by-Step Procedure**

When the event policy invokes an event script, the script output can be written to a file. Configure the event policy **policy1** to write event-script output to a file and upload the generated file to the destination mgmt-archives, which is already configured at the **[edit event-options destinations]** hierarchy level. In this example, the event policy invokes an event script named **event-script1**.

1. Configure the filename of the generated output file.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set event-script event-script1 output-filename policy1-script-output
   ```

2. Configure the **destination** statement to upload the generated file to the desired archive site.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set event-script event-script1 destination mgmt-archives
   ```

3. (Optional) Configure the transfer delay for each destination.

   The following command sets the transfer delay for files uploaded to the mgmt-archives destination to 5 seconds.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# set event-script event-script1 destination mgmt-archives
     transfer-delay 5
   ```

4. (Optional) Configure the retry count and retry interval associated with each destination.

   In this example, if the output file fails to upload to the mgmt-archives site, the system attempts the upload up to 5 more times and waits 4 seconds in between each attempt.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1#set event-script event-script1 destination mgmt-archives retry-count 5
     retry-interval 4
   ```

5. Commit the configuration.

   ```
   [edit event-options policy policy1 then]
   bsmith@R1# commit
   ```

## Results

```
[edit event-options policy policy1 then]
upload filename /var/log/messages destination mgmt-archives {
   user-name admin;
   transfer-delay 4;
   retry-count 5 retry-interval 4;
}
upload filename /var/log/messages destination mgmt-server
upload filename committed destination mgmt-archives;
execute-commands {
   commands {
```

```
        "show interfaces brief ge-*";
      }
      output-filename ge-interfaces;
      destination mgmt-archives {
        transfer-delay 5;
        retry-count 5 retry-interval 4;
      }
    }
    event-script event-script1 {
      output-filename policy1-script-output;
      destination mgmt-archives {
        transfer-delay 5;
        retry-count 5 retry-interval 4;
      }
    }
```

## Verification

### Verifying the Upload

**Purpose** When the configured event triggers the event policy, the system uploads the generated output files and the specified system files to the URL defined in the mgmt-archives destination. On the destination server, verify that all files have been uploaded.

**Action** On the destination server, verify that all uploaded files are present.

```
% ls
R1_ge-interfaces_20111209_213452
R1_juniper.conf.gz_20111209_213409
R1_messages_20111209_212941
R1_policy1-script-output_20111209_212619
```

**Meaning** Note that the filename format for each file includes the device name, the filename, and the date and time stamp.

If all of the uploaded files are present, the event policy and upload actions are working correctly. If none of the files are uploaded, verify that the destination is configured and that the archive site URL and any required password is entered correctly. For information about configuring destinations, see "Example: Defining Destinations for File Archiving by Event Policies" on page 563. If a portion of the files are missing, configure a longer transfer delay and increase the retry count and retry interval for those files.

**Related Documentation**

- Example: Defining Destinations for File Archiving by Event Policies on page 563
- Configuring the Delay Before Files Are Uploaded by an Event Policy on page 572
- Configuring an Event Policy to Retry the File Upload Action on page 574

## Configuring the Delay Before Files Are Uploaded by an Event Policy

A transfer delay allows you to specify the number of seconds the event process (eventd) waits before beginning to upload a file or multiple files. A transfer delay allows you to

ensure that a large file, such as a core file, is completely generated before the upload begins.

As described in "Example: Defining Destinations for File Archiving by Event Policies" on page 563, you can associate a transfer delay with a destination. If you associate a transfer delay with a destination, the transfer delay applies to all file upload actions that use the destination.

In the following example, the ***some-dest*** destination is common for both event policies, **policy1** and **policy2**. A transfer delay of 2 seconds is associated with the **some-dest** destination and applies to uploading the output files to the destination for both event policies.

```
[edit event-options]
policy policy1 {
   events e1;
   then {
   execute-commands {
      commands {
         "show version";
      }
      output-filename command-output.txt;
      destination some-dest;
   }
}
policy policy2 {
   events e2;
   then {
      event-script bar.xsl {
         output-filename event-script-output.txt;
         destination some-dest;
      }
   }
}
destinations {
   some-dest {
      transfer-delay 2;
      archive-sites {
         "scp://robot@my.big.com/foo/moo" password "password";
         "scp://robot@my.little.com/foo/moo" password "password";
      }
   }
}
```

Suppose you have multiple event policy actions that use the same destination. For some of these event policy actions, you want a transfer delay, and for other event policy actions you want no transfer delay. To assign a transfer delay to a single event policy action, include the optional **transfer-delay** statement for each action:

```
transfer-delay seconds;
```

You can include this statement at the following hierarchy levels:

- **[edit** event-options policy *policy-name* then event-script *filename* destination *destination-name*]

- [edit event-options policy *policy-name* then execute-commands destination *destination-name*]
- [edit event-options policy *policy-name* then upload filename (*filename* | committed) destination *destination-name*]

If you configure a transfer delay at the [edit event-options destinations *destination-name*] hierarchy level, and you also configure a transfer delay for the event policy action, the resulting transfer delay is the sum of the two:

```
Total transfer-delay =
transfer-delay (destination) + transfer-delay (event-policy-action)
```

For a configuration example, see "Example: Assigning a Transfer Delay to an Event Policy Action" on page 611.

## Configuring an Event Policy to Retry the File Upload Action

Transient network problems can cause a file upload operation to fail. When this happens, you might want to retry the file upload operation. By default, if the file upload operation fails for any reason, the event policy does not retry the upload operation.

To configure the policy to retry a file upload operation, include the optional **retry-count** and **retry-interval** statements:

```
retry-count number retry-interval seconds;
```

You can include these statements at the following hierarchy levels:

- [edit event-options policy *policy-name* then event-script *filename* destination *destination-name*]
- [edit event-options policy *policy-name* then execute-commands destination *destination-name*]
- [edit event-options policy *policy-name* then upload filename (*filename* | committed) destination *destination-name*]

The **retry-count** statement sets the number of times the policy retries the upload operation if the upload fails. The default value for the **retry-count** statement is 0 and the maximum is 10.

If you include the **retry-count** statement, you can also include the **retry-interval** statement, which sets the time interval (in seconds) between each retry.

For a configuration example, see "Example: Retrying the File Upload Action" on page 618.

## Configuring an Event Policy to Execute Operational Mode Commands

*Operational mode commands* request that the device running Junos OS perform an operation or provide diagnostic output. They allow you to view statistics and information about a device's current operating status. They also allow you to take corrective actions, such as restarting software processes, taking a PIC offline and back online, switching to

redundant interfaces, and adjusting Label Switching Protocol (LSP) bandwidth. For more information about operational mode commands, see the following references:

- Junos OS Operational Mode Commands
- Junos OS Operational Mode Commands
- Junos OS Operational Mode Commands

You can configure a policy that causes operational mode commands to be issued and the output of those commands to be uploaded to a specified location for analysis.

To configure such a policy, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
policy policy-name {
  events [ events ];
  then {
    execute-commands {
      commands {
        "command";
      }
      output-filename filename;
      output-format (text | xml);
      destination destination-name;
    }
  }
}
```

In the **events** statement, you can list multiple events. If one or more of the listed events occurs, the operational mode commands are issued. To view a list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
 <event>
 [              Open a set of values
 acct_accounting_ferror
 acct_accounting_fopen_error
 ...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see "Using Nonstandard System Log Messages to Trigger Event Policies" on page 561.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events to Trigger Event Policies" on page 560.

In the **commands** statement, you can issue multiple operational mode commands upon receipt of a specific event. Enclose each command in quotation marks (" "). The eventd process issues the commands in the order in which they appear in the configuration. For

example, in the following configuration, the execution of **policy1** causes the **show interfaces** command to be issued first, followed by the **show chassis alarms** command:

```
[edit event-options policy policy1 then execute-commands]
user@host# show
commands {
  "show interfaces";
  "show chassis alarms";
}
```

You can include variables in the command to allow data from the triggering event to be automatically included in the command syntax. The eventd process replaces each variable with values contained in the event that triggers the policy. You can use command variables of the following forms:

- **{$$.*attribute-name*}**—The double dollar sign (**$$**) notation represents the event that is triggering a policy. When combined with an attribute name, the variable is replaced by the value of the attribute name in the triggering event. For example, **{$$.interface-name}** stands for the value of the **interface-name** attribute in the triggering event.
- **{$*event*.*attribute-name*}**—The **{$*event*.*attribute-name*}** notation represents the most recent event that matches the specified event. The variable is replaced by the value of the attribute name of the most recent event that matches ***event***. For example, when a policy issues the **show interfaces {$COSD_CHAS_SCHED_MAP_INVALID.interface-name}** command, the **{$COSD_CHAS_SCHED_MAP_INVALID.interface-name}** variable is substituted by the **interface-name** attribute of the most recent **COSD_CHAS_SCHED_MAP_INVALID** event cached by the event process.

For a given event, you can view a list of event attributes that you can reference in an operational mode command by issuing the **help syslog *event-name*** command:

```
user@host> help syslog event-name
```

For example, in the following command output, text in angle brackets (**< >**) shows that **classifier-type** is an attribute of the **cosd_unknown_classifier** event:

```
user@host> help syslog cosd_unknown_classifier
Name:          COSD_UNKNOWN_CLASSIFIER
Message:       rtsock classifier type <classifier-type> is invalid
...
```

You can filter the output of a search by using the pipe (**|**) symbol. The following example lists the filters that can be used with the pipe symbol:

```
user@host# help syslog | ?
Possible completions:
  count               Count occurrences
  display             Show additional kinds of information
  except              Show only text that does not match a pattern
  find                Search for first occurrence of pattern
  hold                Hold text without exiting the --More-- prompt
  last                Display end of output only
  match               Show only text that matches a pattern
```

```
no-more             Don't paginate output
request             Make system-level requests
resolve             Resolve IP addresses
save                Save output text to file
trim                Trim specified number of columns from start of line
```

For more information about using the pipe symbol, see the CLI User Guide.

Another way to view a list of event attributes is to issue the **set attributes-match *event*?** configuration mode command at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
user@host# set attributes-match event ?
```

For example, in the following command output, the ***event.attribute*** list shows that **classifier-type** is an attribute of the **cosd_unknown_classifier** event:

```
[edit event-options policy policy-name]
user@host# set attributes-match cosd_unknown_classifier?
Possible completions:
 <from-event-attribute>  First attribute to compare
 cosd_unknown_classifier.classifier-type
```

NOTE: In this set command, there is no space between the event name and the question mark (?).

To view a list of all event attributes that you can reference, issue the **set attributes-match ?** configuration mode command at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
user@host# set attributes-match ?
Possible completions:
 <from-event-attribute>  First attribute to compare
 acct_accounting_ferror
 acct_accounting_fopen_error
 ...
```

In the **output-filename** statement, assign the name of the file to which to write command output for the specified commands. The filename format is ***hostname_filename_YYYYMMDD_HHMMSS_index-number***.

For each uploaded file, a hostname and timestamp ensure that the uploaded files have unique filenames. If a policy is triggered multiple times in a 1-second period, an index number is added to ensure the filenames are unique. The index number range is 001 through 999.

For example, on a device named **r1**, if you configure the output filename to be **ifl-events**, and this event policy is triggered three times in 1 second, the files are named:

- **r1_ifl-events_20060623_132333**
- **r1_ifl-events_20060623_132333_001**

- r1_ifl-events_20060623_132333_002

By default, the command output format is Junos Extensible Markup Language (XML). To change this, include the **output-format text** statement. This causes the command output to be in formatted ASCII text.

In the **destination** statement, include the destination name that you configured at the **[edit event-options destinations]** hierarchy level. For more information, see "Example: Defining Destinations for File Archiving by Event Policies" on page 563.

For a configuration example, see "Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval" on page 615.

## Executing Event Scripts in an Event Policy

*Event scripts* are Extensible Stylesheet Transformation (XSLT) or Stylesheet Language Alternative Syntax (SLAX) scripts that you write and that are run when triggered by an event policy. Event scripts can perform any function available through Junos XML or Junos XML protocol remote procedure calls (RPCs). Additionally, you can pass to an event script a set of arguments that you define.

A script can change the device configuration, build and run an operational mode command, receive the command output, inspect the output, and determine the next appropriate action. This process can be repeated until the source of the problem is determined. The script can then report the source of the problem to you on the CLI.

You can configure an event policy that causes event scripts to be run and the output of those scripts to be uploaded to a specified location for analysis.

To configure such a policy, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
policy policy-name {
  events [ events ];
  then {
    event-script filename {
      arguments {
        argument-name argument-value;
      }
      output-filename filename;
      output-format (text | xml);
      destination destination-name;
    }
  }
}
```

In the **events** statement, you can list multiple events. If one or more of the listed events occurs, the event script is executed. To view a list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
[edit]
user@host# set event-options policy policy-name events ?
```

```
Possible completions:
  <event>
  [               Open a set of values
  acct_accounting_ferror
  acct_accounting_fopen_error
  ...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see "Using Nonstandard System Log Messages to Trigger Event Policies" on page 561.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events to Trigger Event Policies" on page 560.

In the **event-script** statement, you can specify a script to be executed on receipt of an event. The eventd process runs the scripts in the order in which they appear in the configuration. The scripts that you reference in the **event-script** statement must be located in the **/var/db/scripts/event** directory on the device's hard drive or the **/config/scripts/event/** directory on the flash drive. Furthermore, the event scripts must be enabled at the **[edit event-options event-script file]** hierarchy level. For more information, see "Storing and Enabling Scripts" on page 213.

You can include arguments to the script as name/value pairs. You can include variables in the argument values to allow data from the triggering event to be automatically included in the argument. The eventd process replaces each variable with values contained in the event that triggers the policy. You can use variables of the following forms:

- **{$$.*attribute-name*}**—The double dollar sign (**$$**) notation represents the event that is triggering a policy. When combined with an attribute name, the variable is replaced by the value of the attribute name in the triggering event. For example, **{$$.interface-name}** stands for the value of the **interface-name** attribute in the triggering event.
- **{$*event*.*attribute-name*}**—The **{$*event*.*attribute-name*}** notation represents the most recent event that matches the specified event. The variable is replaced by the value of the attribute name of the most recent event that matches ***event***. For example, when you include an argument called **interface** and define the value as **{$COSD_CHAS_SCHED_MAP_INVALID.interface-name},** the **{$COSD_CHAS_SCHED_MAP_INVALID.interface-name}** variable is replaced by the **interface-name** attribute of the most recent **COSD_CHAS_SCHED_MAP_INVALID** event cached by the eventd process.

For a given event, you can view a list of event attributes that you can reference by issuing the **help syslog *event*** command:

```
user@host> help syslog event-name
```

For example, in the following command output, text in angle brackets (**< >**) shows attributes of the **COSD_CHASSIS_SCHEDULER_MAP_INVALID** event:

```
user@host> help syslog COSD_CHASSIS_SCHEDULER_MAP_INVALID
Name:        COSD_CHASSIS_SCHEDULER_MAP_INVALID
```

```
Message:     Chassis scheduler map incorrectly applied to interface
<interface-name>: <error-message>
...
```

You can filter the output of a search by using the pipe (**|**) symbol. The following example lists the filters that can be used with the pipe symbol:

```
user@host> help syslog | ?
Possible completions:
 count               Count occurrences
 display             Show additional kinds of information
 except              Show only text that does not match a pattern
 find                Search for first occurrence of pattern
 hold                Hold text without exiting the --More-- prompt
 last                Display end of output only
 match               Show only text that matches a pattern
 no-more             Don't paginate output
 request             Make system-level requests
 resolve             Resolve IP addresses
 save                Save output text to file
 trim                Trim specified number of columns from start of line
```

For more information about using the pipe symbol, see the CLI User Guide.

Another way to view a list of event attributes is to issue the **set attributes-match *event* ?** configuration mode command at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
user@host# set attributes-match event ?
```

For example, in the following command output, the ***event.attribute*** list shows that **error-message** and **interface-name** are attributes of the **cosd_chassis_scheduler_map_invalid** event:

```
[edit event-options policy p1]
user@host# set attributes-match cosd_chassis_scheduler_map_invalid?
Possible completions:
  <from-event-attribute>  First attribute to compare
  cosd_chassis_scheduler_map_invalid.error-message
  cosd_chassis_scheduler_map_invalid.interface-name
```

In this **set** command, there is no space between the event name and the question mark (**?**).

To view a list of all event attributes that you can reference, issue the **set attributes-match ?** configuration mode command at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
user@host# set attributes-match ?
Possible completions:
  <from-event-attribute>  First attribute to compare
  acct_accounting_ferror
  acct_accounting_fopen_error
  ...
```

By default, the command output format is text. To change this, include the **output-format xml** statement.

In the optional **output-filename** statement, assign the name of the file to which to write script output for the specified script.

The filename format is ***hostname_filename_YYYYMMDD_HHMMSS_index-number***.

For each uploaded file, a hostname and timestamp are automatically added to the filename to ensure that the uploaded files have unique filenames. If a policy is triggered multiple times in a 1-second period, an index number is added to ensure the filenames are unique. The index number range is 001 through 999.

For example, on a device named **r1**, if you configure the output filename to be **ifl-events**, and this event policy is triggered three times in 1 second, the files are named:

- **r1_ifl-events_20060623_132333**
- **r1_ifl-events_20060623_132333_001**
- **r1_ifl-events_20060623_132333_002**

In the optional **destination** statement, include the destination name that you configured at the **[edit event-options destinations]** hierarchy level. For more information, see "Example: Defining Destinations for File Archiving by Event Policies" on page 563.

For the **output-filename** and **destination** statements, there are four configuration scenarios:

- You can omit the **output-filename** and **destination** statements. This option makes sense when the event script has no output. For example, the event script might execute only **request** commands, which have no output.
- You can include the **destination** statement in the configuration. You omit the **output-filename** statement in the configuration and specify an output filename in the event script instead. The script output is sent to the destination specified in the configuration. If you do not include the **destination** statement in the configuration, the script output is not uploaded.

  In this scenario, the event policy extracts the filename from the event script. The event script writes the output filename as **STDOUT**. The XML syntax to use in the event script is:

  ```
  <output>
    <event-script-output-filename>filename</event-script-output-filename>
  </output>
  ```

  The **<event-script-output-filename>** element must be the first child tag within the **<output>** parent tag.

  On a device named **device2**, configure an event script action with a destination **host**, and omit the **output-filename** statement. Define the destination **host** as ftp://user@device1//tmp.

  In the **script1.xsl** event script, write the following output to **STDOUT**:

  ```
  <event-script-output-filename>/var/cmd.txt</event-script-output-filename>
  ```

  Configure the **policy1** event policy as follows:

  ```
  [edit event-options]
  ```

```
policy policy1 {
  then {
    event-script script1.xsl {
      destination host;
    }
  }
}
destinations {
  host {
    archive-sites {
      "ftp://user@device1//tmp" password "$9$xkJNbYg4ZDH.oJ.fQnpuSyl";  ##
        SECRET-DATA***
    }
  }
}
```

In this example, the **/var/cmd.txt** file resides on device **device2**. The event policy uses the File Transfer Protocol (FTP) to upload this file to the **/tmp** directory on device **device1**.

The event policy reads the output filename **/var/cmd.txt** from **STDOUT**. Then the event policy uploads the **/var/cmd.txt** file to the configured destination, which is the **/tmp** directory on device **device1**. The event policy renames the **/var/cmd.txt** file as **device2_cmd.txt_*YYYYMMDD_HHMMSS_range***.

- You can include the **output-filename** and **destination** statements. If you include the **output-filename** statement in the configuration, you must also include the **destination** statement in the configuration. In this case, the script output is redirected to the output filename specified in the configuration and is sent to the destination specified in the configuration.
- You can include the **output-filename** and **destination** statements, and also specify an output filename directly within the event script. If you do this, the output filename specified in the configuration overrides the output filename specified in the event script.

## Configuring Event Policies to Ignore an Event

You can modify a policy to cause particular events to be ignored or to cause all events to be ignored during a particular time interval, to allow for maintenance for example. To configure such a policy, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
policy policy-name {
  events [ events ];
  then {
    ignore;
  }
}
```

In the **events** statement, you can list multiple events. To view a list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
[edit]
```

```
user@host# set event-options policy policy-name events ?
Possible completions:
 <event>
 [              Open a set of values
 acct_accounting_ferror
 acct_accounting_fopen_error
 ...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see "Using Nonstandard System Log Messages to Trigger Event Policies" on page 561.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events to Trigger Event Policies" on page 560.

If one or more of the listed events occur, a system log message for the event is not generated, and no further policies associated with this event are processed. If you include the **ignore** statement in a policy configuration, you cannot configure any other actions in the policy.

## Changing the User Privilege Level for an Event Policy Action

Only superusers can configure event policies. Event policy actions—such as executing event scripts, uploading files, and executing operational mode commands—are by default executed by user **root**, because the event process (eventd) runs with **root** privileges.

In some cases, you might want an event policy action to be executed with restricted privileges. For example, suppose you configure an event policy that executes a script if an interface goes down. The script includes remote procedure calls (RPCs) to change the device configuration if certain conditions are present. If you do not want the script to change the configuration, you can execute the script with a restricted user profile. When the script is executed with a user profile that disallows configuration changes, the RPCs to change the configuration fail.

You can associate a user with each action in an event policy. If a user is not associated with an event policy action, then the action is executed as user **root** by default.

To specify the user under whose privileges an action is executed, include the **user-name** statement:

```
user-name username;
```

You can include this statement at the following hierarchy levels:

- **[edit** event-options policy ***policy-name*** then event-script ***filename*****]**
- **[edit** event-options policy ***policy-name*** then execute-commands**]**
- **[edit** event-options policy ***policy-name*** then upload **filename (***filename* **| committed) destination** ***destination-name*****]**

NOTE: The username that you specify must be configured at the **[edit system login]** hierarchy level. For more information, see the Junos OS System Basics Configuration Guide.

For a configuration example, see "Example: Associating an Optional User with an Event Policy Action" on page 613.

## Example: Configuring Event Policies to Raise SNMP Traps



### Overview of Using Event Policies to Raise SNMP Traps

SNMP *traps* enable an agent to notify a network management system (NMS) of significant events by way of an unsolicited SNMP message. You can configure an event policy action that raises traps for events based on system log messages. If one or more of the listed events occur, the system log message for the event is converted into a trap. This enables notification of an SNMP trap-based application when an important system log message occurs. You can convert any system log message (for which there are no corresponding traps) into a trap. This is helpful if you use NMS traps rather than system log messages to monitor your network.

To configure an event policy that raises a trap on receipt of an event, include the following statements at the **[edit event-options policy *policy-name*]** hierarchy level:

```
[edit event-options policy policy-name]
events [ events ];
then {
  raise-trap;
}
```

The Juniper Networks enterprise-specific System Log MIB, whose object identifier is {jnxMibs 35}, provides support for this feature.

### Example: Raising an SNMP Trap in Response to an Event

This example configures an event policy to raise a trap and to execute an event script in response to an event:



#### Requirements

A device running Junos OS, which is configured for SNMP.

## Overview

The following example configures the event policy **raise-trap-on-ospf-nbrdown** to trigger on the **RPD_OSPF_NBRDOWN** event, which indicates a terminated OSPF adjacency with a neighboring router. The event policy action raises a trap in response to the event. The device sends a notification to the SNMP manager, if one is configured under the **[edit snmp]** hierarchy level.

Additionally, the event policy executes the event script **ospf.xsl** in response to this event and provides the affected interface as an argument to the script. The **$$rpd_ospf_nbrdown.interface-name** argument resolves to the interface name associated with the triggering event.

The event script output is recorded in the file **ospf-out**, and the output file is uploaded to the destination **mgmt-archives**, which is configured at the **[edit event-options destinations]** hierarchy level. To invoke an event script in an event policy, the event script must be present in the **/var/db/scripts/event** directory on the hard drive, and it must be enabled in the configuration.

## Configuration

**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level.

```
set event-options policy raise-trap-on-ospf-nbrdown events rpd_ospf_nbrdown
set event-options policy raise-trap-on-ospf-nbrdown then raise-trap
set event-options policy raise-trap-on-ospf-nbrdown then event-script ospf.xsl arguments
  interface "{$$rpd_ospf_nbrdown.interface-name}"
set event-options policy raise-trap-on-ospf-nbrdown then event-script ospf.xsl
  output-filename ospf-out
set event-options policy raise-trap-on-ospf-nbrdown then event-script ospf.xsl destination
  mgmt-archives
```

### *Configuring the Event Policy*

**Step-by-Step Procedure**

To configure an event policy that raises a trap on receipt of an event and optionally executes an event script:

1. Create and name the event-policy.

   ```
   [edit]
   user@R1# edit event-options policy raise-trap-on-ospf-nbrdown
   ```

2. Configure the event policy to match on the desired event, which in this example is the **RPD_OSPF_NBRDOWN** event.

   ```
   [edit event-options policy raise-trap-on-ospf-nbrdown]
   user@R1# set events rpd_ospf_nbrdown
   ```

3. Configure the event policy action to raise an SNMP trap in response to the event.

   ```
   [edit event-options policy raise-trap-on-ospf-nbrdown]
   user@R1# set then raise-trap
   ```

4. (Optional) Configure additional actions to take in response to the event.

   This example executes an event script and uploads the associated output file to a predefined destination.

   ```
   [edit event-options policy raise-trap-on-ospf-nbrdown]
   user@R1# set then event-script ospf.xsl arguments interface
     {$$rpd_ospf_nbrdown.interface-name}
   user@R1# set then event-script ospf.xsl output-filename ospf-out destination
     mgmt-archives
   ```

5. Commit the configuration.

   ```
   user@R1# commit
   ```

**Results**

```
[edit event-options]
policy raise-trap-on-ospf-nbrdown {
  events rpd_ospf_nbrdown;
  then {
    event-script ospf.xsl {
      arguments {
        interface "{$$rpd_ospf_nbrdown.interface-name}";
      }
      output-filename ospf-out;
      destination mgmt-archives;
    }
    raise-trap;
  }
}
```

**Related Documentation**

- Configuring SNMP on a Device Running Junos OS
- Interpreting the Enterprise-Specific System Log MIB

## Tracing Event Policy Processing

Event policy tracing operations track all event policy operations and record them in a log file. The logged error descriptions provide detailed information to help you solve problems faster.

By default, no events are traced. If you include the **traceoptions** statement at the **[edit event-options]** hierarchy level, the default tracing behavior is the following:

- Important events are logged in a file called **eventd** located in the **/var/log** directory.
- When the file **eventd** reaches 128 kilobytes (KB), it is renamed and compressed to **eventd.0.gz**, then **eventd.1.gz**, and so on, until there are three trace files. Then the oldest trace file (**eventd.2.gz**) is overwritten. (For more information about how log files are created, see the *Junos OS System Log Messages Reference*.)
- Log files can be accessed only by the user who configures the tracing operation.

You cannot change the directory (**/var/log**) to which trace files are written. However, you can customize the other trace file settings by including the following statements at the **[edit event-options traceoptions]** hierarchy level:

```
[edit event-options traceoptions]
file <filename> <files number> <match regular-expression> <size size> <world-readable |
  no-world-readable>;
flag all;
flag configuration;
flag database;
flag events;
flag policy;
flag server;
flag syslog
flag timer-events;
no-remote-trace;
```

These statements are described in the following sections:

- Configuring the Event Policy Log Filename on page 587
- Configuring the Number and Size of Event Policy Log Files on page 587
- Configuring Access to the Log File on page 588
- Configuring a Regular Expression for Lines to Be Logged on page 588
- Configuring the Trace Operations on page 588

## Configuring the Event Policy Log Filename

By default, the name of the file that records trace output is **eventd**. You can specify a different name by including the **file** statement at the **[edit event-options traceoptions]** hierarchy level:

```
[edit event-options traceoptions]
file filename;
```

## Configuring the Number and Size of Event Policy Log Files

By default, when the trace file reaches 128 kilobytes (KB) in size, it is renamed ***filename*.0**, then ***filename*.1**, and so on, until there are three trace files. Then the oldest trace file (***filename*.2**) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the **[edit event-options traceoptions file <*filename*>]** hierarchy level:

```
[edit event-options traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 2 MB and the maximum number of files to 20. When the file that receives the output of the tracing operation (***filename***) reaches 2 MB, ***filename*** is renamed and compressed to ***filename*.0.gz** and a new file called ***filename*** is created.

When ***filename*** reaches 2 MB, ***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).

## Configuring Access to the Log File

By default, log files can be accessed only by the user who configures the tracing operation.

To specify that any user can read all log files, include the **world-readable** statement at the **[edit event-options traceoptions file <*filename*>]** hierarchy level:

```
[edit event-options traceoptions file <filename>]
world-readable;
```

To explicitly set the default behavior, include the **no-world-readable** statement at the **[edit event-options traceoptions file <*filename*>]** hierarchy level:

```
[edit event-options traceoptions file <filename>]
no-world-readable;
```

## Configuring a Regular Expression for Lines to Be Logged

By default, the trace operation output includes all lines relevant to the logged events.

You can refine the output by including the **match** statement at the **[edit event-options traceoptions file <*filename*>]** hierarchy level and specifying a regular expression to be matched:

```
[edit event-options traceoptions file <filename>]
match regular-expression;
```

## Configuring the 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 event-options traceoptions]** hierarchy level:

```
[edit event-options traceoptions]
flag all;
flag configuration;
flag database;
flag events;
flag policy;
flag server;
flag syslog
flag timer-events;
```

Table 40 on page 588 describes the meaning of the event policy tracing flags.

### Table 40: Event Policy Tracing Flags

| Flag | Description | Default Setting |
|---|---|---|
| **all** | Trace all operations. | Off |
| **configuration** | Log reading of configuration at the **[edit event-options]** hierarchy level. | Off |

Table 40: Event Policy Tracing Flags *(continued)*

| Flag | Description | Default Setting |
|---|---|---|
| **events** | Trace important events. | Off |
| **database** | Log events involving storage and retrieval in events database. | Off |
| **policy** | Log policy processing. | Off |
| **server** | Log communication with processes that are generating events. | Off |
| **syslogd** | Log syslog related traces | Off |
| **timer-events** | Log internally generated events. | Off |

To display the end of the log, issue the **show log eventd | last** operational mode command:

```
[edit]
user@host# run show log eventd | last
```

**Related Documentation**

- Configuring the System Log Priority of the Triggering Event in an Event Policy on page 589

## Configuring the System Log Priority of the Triggering Event in an Event Policy

- Understanding the Event System Log Priority in an Event Policy on page 589
- Example: Configuring the Event System Log Priority in an Event Policy on page 590

### Understanding the Event System Log Priority in an Event Policy

Starting with Junos OS Release 12.1, you can configure an event policy to override the default system log priority of a triggering event so that the system logs the event with a different facility type, severity level, or both. To override the priority of the triggering event, configure the **priority-override** statement at the **[edit** event-options policy ***policy-name*** then**]** hierarchy level. To override the facility type with which the triggering event is logged, include the **facility** statement and the new facility type. To override the severity level with which the triggering event is logged, include the **severity** statement and the new severity level.

Junos OS processes generate system log messages, or event notifications, to record the events that occur on a routing, switching, or security platform. Each system log message identifies the Junos OS process that generated the message and describes the operation or error that occurred. The Junos OS event process (eventd) receives the event notifications, and configured event policies instruct the eventd process to perform a set of actions upon receipt of specific events or correlated events.

Each system log message belongs to a facility, which groups messages that either are generated by the same source (such as a software process) or concern a similar condition

or activity (such as authentication attempts). Each message is also preassigned a severity level, which indicates how seriously the triggering event affects the functions of the routing, switching, or security platform. A message's facility and severity level are together referred to as its priority. For more information about facility and severity levels, see Junos OS System Logging Facilities and Message Severity Levels.

When you configure logging on a device for a specific facility and destination, you also specify a severity level. Messages from that facility that are rated at the configured severity level or higher are logged. To log related events with different severity levels in the same log file, you must filter events using the lowest severity level of any of the events from that facility to be logged. This can result in unwieldy log files that are difficult and time-consuming to parse.

For example, Junos OS logs the protocol UP and DOWN events with different severity levels. Both the SNMP_TRAP_LINK_DOWN and SNMP_TRAP_LINK_UP events have a facility of 'daemon', but the SNMP_TRAP_LINK_DOWN event has a severity level of 'warning', and the SNMP_TRAP_LINK_UP event has a severity level of 'info'. Normally, when you configure a system log file, you must filter events to that file using the lower severity level of 'info' in order to log both of the events.

The event policy **priority-override** statement enables you to customize the priority of the triggering event so that it is logged using a different facility type and severity level. Suppose you configure a system log file to filter events of facility 'daemon' and severity 'notice', and you have event policies that trigger on the RPD_ISIS_ADJDOWN and RPD_ISIS_ADJUP events. When the system generates an RPD_ISIS_ADJDOWN message reporting that the IS-IS adjacency with a neighboring router was terminated, this message is logged. However, if the system subsequently generates an RPD_ISIS_ADJUP event notification reporting that the IS-IS adjacency has been restored, by default, the message is not logged, because it has a lower severity level of 'info'. In the event policy that triggers on the RPD_ISIS_ADJUP event, you can configure the associated priority so that the triggering RPD_ISIS_ADJUP event is logged with a severity level of 'notice' and is captured in the configured log file.

NOTE: Event policies are executed in the order in which they appear in the configuration. When you configure multiple event policies to override the priority of the same event, the event is logged based on the priority set by the last executed event policy to change it.

## Example: Configuring the Event System Log Priority in an Event Policy

It is necessary to log events when monitoring, managing, and troubleshooting routing, switching, and security devices. Starting with Junos OS Release 12.1, you can configure an event policy to override the priority of its triggering event so that it is logged based on a different facility type and severity level. This enables the event to be logged even if the system filters events to the destination log file using a different facility type or a higher severity level.

This example simulates an SNMP_TRAP_LINK_UP event for a specific interface. Upon receipt of the event, the event policy overrides the severity level of the event so that it is captured in the configured log file.



## Requirements

- Routing, switching, or security device running Junos OS Release 12.1 or later.
- Interface is configured and active. This example uses the ge-0/3/1.0 interface.

## Overview

This example configures two log files to capture events of facility 'daemon'. One log file is configured to filter for events of severity 'warning' or higher, and the second log file is configured to filter for events of severity 'info' or higher.

The configured event policy triggers on the SNMP_TRAP_LINK_UP event for interface ge-0/3/1.0. The example generates an SNMP_TRAP_LINK_DOWN event followed by an SNMP_TRAP_LINK_UP event for the ge-0/3/1.0 interface. The SNMP_TRAP_LINK_DOWN event, which has a severity level of 'warning' is captured in both configured log files. Upon receipt of the SNMP_TRAP_LINK_UP event, the event policy overrides the severity level of the event to 'warning' so that it is also captured in the log file that filters for events of severity 'warning'. By default, if the event policy does not override the severity level of this event, it is only captured in the log file that filters for the severity level 'info'.

## Configuration



**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set system syslog file syslog-event-daemon-info daemon info
set system syslog file syslog-event-daemon-warning daemon warning
set event-options policy log-on-snmp-trap-link-up events snmp_trap_link_up
set event-options policy log-on-snmp-trap-link-up attributes-match
   snmp_trap_link_down.interface-name matches ge-0/3/1.0
set event-options policy log-on-snmp-trap-link-up then priority-override severity warning
```

*Configuring the Log Files*

**Step-by-Step Procedure**

1. Configure two log files at the **[edit system syslog]** hierarchy level to record events of facility **daemon**.

   Configure one log to record events of severity 'info' or higher and one log file to record events of severity 'warning' or higher.

   ```
   [edit system syslog]
   bsmith@R1# set file syslog-event-daemon-info daemon info
   bsmith@R1# set file syslog-event-daemon-warning daemon warning
   ```

2. Commit the configuration.

   ```
   bsmith@R1# commit
   ```

3. To manually test the logging of the events, take the ge-0/3/1.0 logical interface temporarily offline, and then bring it back up.

   This generates an SNMP_TRAP_LINK_DOWN event followed by an SNMP_TRAP_LINK_UP event.

   ```
   [edit]
   bsmith@R1# set interfaces ge-0/3/1 unit 0 disable
   bsmith@R1# commit
   bsmith@R1# delete interfaces ge-0/3/1 unit 0 disable
   bsmith@R1# commit
   ```

**Results**

```
[edit]
system {
   syslog {
      file syslog-event-daemon-info {
         daemon info;
      }
      file syslog-event-daemon-warning {
         daemon warning;
      }
   }
}
```

*Verifying the Default System Log Priority of the Events*

**Purpose** Verify that the system generated the SNMP_TRAP_LINK_DOWN and SNMP_TRAP_LINK_UP events for the ge-0/3/1.0 interface, and note where each event is logged.

**Action** Review the contents of the **syslog-event-daemon-info** file configured in Step 1 of the previous procedure. The output shows that the ge-0/3/1.0 interface was brought down and back up and generated an SNMP_TRAP_LINK_DOWN event followed by an SNMP_TRAP_LINK_UP event.

```
bsmith@R1> show log syslog-event-daemon-info
Oct 24 13:22:17  R1 mib2d[1394]: SNMP_TRAP_LINK_DOWN: ifIndex 539, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
...
Oct 24 13:22:29  R1 mib2d[1394]: SNMP_TRAP_LINK_UP: ifIndex 539, ifAdminStatus
up(1), ifOperStatus up(1), ifName ge-0/3/1.0
```

Review the contents of the **syslog-event-daemon-warning** file configured in Step 1 of the previous procedure. Because the severity level of the SNMP_TRAP_LINK_UP event is 'info', it does not appear in a log file that is configured to only record events of severity 'warning' or higher. By default, this system log file captures the SNMP_TRAP_LINK_DOWN events, but does not capture the SNMP_TRAP_LINK_UP events.

```
bsmith@R1> show log syslog-event-daemon-warning
Oct 24 13:22:17  R1 mib2d[1394]: SNMP_TRAP_LINK_DOWN: ifIndex 539, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
```

**Meaning** Because the SNMP_TRAP_LINK_UP event has a default severity of 'info', it is not forwarded to log files that are configured to capture events of higher severity.

*Configuring the Event Policy*

**Step-by-Step Procedure**

1. Create and name the event-policy.

   ```
   [edit]
   bsmith@R1# edit event-options policy log-on-snmp-trap-link-up
   ```

2. Configure the **events** statement.

   For this example, the event policy triggers on the SNMP_TRAP_LINK_UP event. Set the **attributes-match** statement so that the policy triggers only if the SNMP_TRAP_LINK_UP event occurs for the ge-0/3/1.0 interface.

   ```
   [edit event-options policy log-on-snmp-trap-link-up]
   bsmith@R1# set events snmp_trap_link_up
   bsmith@R1# set attributes-match snmp_trap_link_down.interface-name matches
     ge-0/3/1.0
   ```

3. Configure the **priority-override** event policy action, and include the **severity** statement with a value of **warning**.

   ```
   [edit event-options policy log-on-snmp-trap-link-up]
   bsmith@R1# set then priority-override severity warning
   ```

4. Commit the configuration.

   ```
   bsmith@R1# commit
   ```

5. To manually test the event policy, take the ge-0/3/1.0 logical interface temporarily offline, and then bring it back up. This generates an SNMP_TRAP_LINK_DOWN event followed by an SNMP_TRAP_LINK_UP event.

   ```
   [edit]
   ```

```
bsmith@R1# set interfaces ge-0/3/1 unit 0 disable
bsmith@R1# commit
bsmith@R1# delete interfaces ge-0/3/1 unit 0 disable
bsmith@R1# commit
```

**Results**

```
[edit]
event-options {
  policy log-on-snmp-trap-link-up {
    events snmp_trap_link_up;
    attributes-match {
      snmp_trap_link_up.interface-name matches ge-0/3/1.0;
    }
    then {
      priority-override {
        severity warning;
      }
    }
  }
}
```

## Verification

Confirm that the configuration is working properly.

***Verifying the Configured System Log Priority of the Events***

**Purpose** Verify that the system generated the SNMP_TRAP_LINK_DOWN and SNMP_TRAP_LINK_UP events for the ge-0/3/1.0 interface, and note where each event is logged.

**Action** Review the contents of the **syslog-event-daemon-warning** file. Because the event policy overrides the severity level of the SNMP_TRAP_LINK_UP event, it now appears in the log file that is configured to only record events of severity 'warning' or higher. By default, this system log file captures the SNMP_TRAP_LINK_DOWN events, but does not capture the SNMP_TRAP_LINK_UP events.

```
bsmith@R1> show log syslog-event-daemon-warning
Oct 24 13:29:48  R1 mib2d[1394]: SNMP_TRAP_LINK_DOWN: ifIndex 539, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
Oct 24 13:30:02  R1 mib2d[1394]: SNMP_TRAP_LINK_UP: ifIndex 539, ifAdminStatus
up(1), ifOperStatus up(1), ifName ge-0/3/1.0
```

**Meaning** Although the SNMP_TRAP_LINK_UP event has a severity of 'info', configuring the **priority-override** statement with a severity of 'warning' causes the event to be forwarded to the system logs with the configured severity level. The event can be captured in logs that filter for a different facility type and a higher severity level.

**Related Documentation**

- Junos OS System Logging Facilities and Message Severity Levels
- Specifying the Facility and Severity of Messages to Include in the Log
- facility on page 638
- priority-override on page 645

- severity on page 648

## Configuring an Event Policy to Change the Configuration



### Configuring an Event Policy to Change the Configuration Overview

An event policy performs actions in response to specific events. You can configure custom event policies in the Junos OS configuration that listen for a specific event or correlated events and then execute an action, which might include creating a log file, invoking Junos OS commands, or executing an event script.

At times, it might be necessary to modify the configuration in response to a particular event. Prior to Junos OS Release 12.1, an event policy invoked an event script to execute configuration changes. Starting with Junos OS Release 12.1, you can configure an event policy to modify the configuration using Junos OS configuration mode commands and then commit the updated configuration. For example, for an SNMP_TRAP_LINK_DOWN or SNMP_TRAP_LINK_UP event for a given interface, the event policy action might modify the configuration of a static route to adjust its metric or modify its next hop.

You configure event policy actions at the **[edit event-options policy *policy-name* then]** hierarchy level. To modify the configuration through an event policy, configure the **change-configuration** statement and specify the configuration mode commands that are executed upon receipt of the configured event or events. Enclose each command in quotation marks (**" "**), and specify the complete statement path to the element, identifier, or value as you do in configuration mode when issuing commands at the **[edit]** hierarchy level.

The event process (eventd) executes the configuration commands in the order in which they appear in the event policy configuration. The commands update the candidate configuration, which is then committed, provided that no commit errors occur.

You can configure the **commit-options** child statement to customize the event policy commit operation. You can commit the changes on a single Routing Engine or configure the **synchronize** option to synchronize the commit on both Routing Engines. The Routing Engine on which you execute this command copies and loads its candidate configuration to the other Routing Engine. Both Routing Engines perform a syntax check on the candidate configuration file. If no errors are found, the configuration is activated and becomes the current operational configuration on both Routing Engines. By default, the **synchronize** option does not work if the responding Routing Engine has uncommitted configuration changes. However, you can enforce commit synchronization on the Routing Engines and ignore any warnings by configuring the **force** option.

Additionally, if you are testing or troubleshooting an event policy, you can configure the **check** commit option to verify the candidate configuration syntax without committing the changes. On dual control plane systems, when the **check synchronize** statement is

configured, the candidate configuration on one control plane is copied to the other control plane, and the system verifies that both candidate configurations are syntactically correct. The **check** statement and the other **commit-options** statements are mutually exclusive.

The change configuration action might fail while acquiring a lock on the configuration. Configure the **retry** statement to have the system attempt the change configuration event policy action a specified number of times if the first attempt fails. Configure the **user-name** statement to execute the configuration changes and commit under the privileges of a specific user. If you do not specify a username, the action is executed as user **root**.

## Example: Changing the Configuration Using an Event Policy

It might be necessary to modify the configuration in response to a particular event. Starting with Junos OS Release 12.1, you can configure an event policy to make and commit configuration changes when the event policy is triggered by one or more specific events.

This example simulates an SNMP_TRAP_LINK_DOWN event for a specific interface. Upon receipt of the event, the event policy modifies the configuration of a static route to use a new next-hop IP address through a different exit interface.



### Requirements

- Routing, switching, or security device running Junos OS Release 12.1 or later.

### Overview

You can configure an event policy action to modify the configuration when the policy is triggered by a single event or correlated events. Suppose you have a static route to the 10.1.10.0/24 network with a next-hop IP address of 10.1.2.1 through the exit interface ge-0/3/1. At some point, this interface goes down, triggering an SNMP_TRAP_LINK_DOWN event.

This example creates an event policy named update-on-snmp-trap-link-down. The event policy is configured so that the eventd process listens for an SNMP_TRAP_LINK_DOWN event associated with the interface ge-0/3/1.0. If the interface goes down, the event policy executes a change configuration action. Commands are executed in the order in which they appear in the event policy. The event policy configuration commands remove the static route through the ge-0/3/1 exit interface and create a new static route to the same target network with a next-hop IP address of 10.1.3.1 through the exit interface ge-0/2/1.

The event policy change configuration commit operation is executed under the username bsmith with a commit comment specifying that the change was made through the

associated event policy. The retry count is set to 5 and the retry interval is set to 4 seconds. If the initial attempt to issue the configuration change fails, the system attempts the configuration change 5 additional times and waits 4 seconds between each attempt.

Although not presented here, you might have a second, similar event policy that executes a change configuration action to update the static route when the interface comes back up. In that case the policy would trigger on the SNMP_TRAP_LINK_UP event for the same interface.

## Configuration

**CLI Quick Configuration**

To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set event-options policy update-on-snmp-trap-link-down events snmp_trap_link_down
set event-options policy update-on-snmp-trap-link-down attributes-match
    snmp_trap_link_down.interface-name matches ge-0/3/1.0
set event-options policy update-on-snmp-trap-link-down then change-configuration
    retry count 5
set event-options policy update-on-snmp-trap-link-down then change-configuration
    retry interval 4
set event-options policy update-on-snmp-trap-link-down then change-configuration
    commands "delete routing-options static route 10.1.10.0/24 next-hop"
set event-options policy update-on-snmp-trap-link-down then change-configuration
    commands "set routing-options static route 10.1.10.0/24 next-hop 10.1.3.1"
set event-options policy update-on-snmp-trap-link-down then change-configuration
    user-name bsmith
set event-options policy update-on-snmp-trap-link-down then change-configuration
    commit-options log "updating configuration from event policy
    update-on-snmp-trap-link-down"
set routing-options static route 10.1.10.0/24 next-hop 10.1.2.1
set system syslog file syslog-event-daemon-warning daemon warning
```

*Configuring the Event Policy*

**Step-by-Step Procedure**

1. Create and name the event policy.

   ```
   [edit]
   bsmith@R1# edit event-options policy update-on-snmp-trap-link-down
   ```

2. Configure the **events** statement so that the event policy triggers on the SNMP_TRAP_LINK_DOWN event.

   Set the **attributes-match** statement so that the policy triggers only if the SNMP_TRAP_LINK_DOWN event occurs for the ge-0/3/1.0 interface.

   ```
   [edit event-options policy update-on-snmp-trap-link-down]
   bsmith@R1# set events snmp_trap_link_down
   bsmith@R1# set attributes-match snmp_trap_link_down.interface-name matches
     ge-0/3/1.0
   ```

3. Specify the configuration mode commands that are executed if the ge-0/3/1 interface goes down.

   Configure each command on a single line, enclose the command string in quotes, and specify the complete statement path.

   ```
   [edit event-options policy update-on-snmp-trap-link-down then
     change-configuration]
   bsmith@R1# set commands "delete routing-options static route 10.1.10.0/24
     next-hop"
   bsmith@R1# set commands "set routing-options static route 10.1.10.0/24 next-hop
     10.1.3.1"
   ```

4. Configure the commit options.

   Configure the **log** option with a comment describing the configuration changes. The comment is added to the commit logs after a successful commit operation is made through the associated event policy.

   ```
   [edit event-options policy update-on-snmp-trap-link-down then
     change-configuration]
   bsmith@R1# set commit-options log "updating configuration from event policy
     update-on-snmp-trap-link-down"
   ```

   If you have dual Routing Engines, configure the **synchronize** option to commit the configuration on both Routing Engines. Include the **force** option to force the commit on the other Routing Engine, ignoring any warnings. This example does not configure the **synchronize** and **force** options.

5. (Optional) Configure the retry count and retry interval.

   In this example, **count** is set to 5 and the **interval** is 4 seconds.

   ```
   [edit event-options policy update-on-snmp-trap-link-down then
     change-configuration]
   bsmith@R1# set retry count 5 interval 4
   ```

6. (Optional) Configure the username under whose privileges the configuration changes and commit are made.

   If you do not specify a username, the action is executed as user **root**.

   ```
   [edit event-options policy update-on-snmp-trap-link-down then
     change-configuration]
   bsmith@R1# set user-name bsmith
   ```

7. Configure a new log file at the **[edit system syslog]** hierarchy level to record syslog events of facility **daemon** and severity **warning**.

   This captures the SNMP_TRAP_LINK_DOWN events.

   ```
   [edit system syslog]
   bsmith@R1# set file syslog-event-daemon-warning daemon warning
   ```

8. To test this example, configure a static route to the 10.1.10.0/24 network with a next hop IP address of 10.1.2.1.

   ```
   [edit]
   bsmith@R1# set routing-options static route 10.1.10.0/24 next-hop 10.1.2.1
   ```

9. Commit the configuration.

```
bsmith@R1# commit
```

10. Review the **[edit routing-options static]** hierarchy level of the configuration before disabling the ge-0/3/1 interface, and note the next hop IP address.

```
bsmith@R1> show configuration routing-options static
...
route 10.1.10.0/24 next-hop 10.1.2.1;
...
```

11. To manually test the event policy, take the ge-0/3/1 interface temporarily offline to generate the SNMP_TRAP_LINK_DOWN event.

```
[edit]
bsmith@R1# set interfaces ge-0/3/1 disable
bsmith@R1# commit
```

***Results***

```
[edit]
event-options {
    policy update-on-snmp-trap-link-down {
        events snmp_trap_link_down;
        attributes-match {
            snmp_trap_link_down.interface-name matches ge-0/3/1.0;
        }
        then {
            change-configuration {
                retry count 5 interval 4;
                commands {
                    "delete routing-options static route 10.1.10.0/24 next-hop";
                    "set routing-options static route 10.1.10.0/24 next-hop 10.1.3.1";
                }
                user-name bsmith;
                commit-options {
                    log "updating configuration from event policy update-on-snmp-trap-link-down";
                }
            }
        }
    }
}
routing-options {
    static {
        route 10.1.10.0/24 next-hop 10.1.2.1;
    }
}
system {
    syslog {
        file syslog-event-daemon-warning {
            daemon warning;
        }
    }
}
```

## Verification

Confirm that the configuration is working properly.



### *Verifying the Status of the Interface*

**Purpose** Verify that the ge-0/3/1 interface is down and that it triggered the SNMP_TRAP_LINK_DOWN event.

**Action** Issue the **show interfaces ge-0/3/1** operational mode command. The command output shows that the interface is administratively offline.

```
bsmith@R1> show interfaces ge-0/3/1
Physical interface: ge-0/3/1, Administratively down, Physical link is Down
<output omitted>
```

Review the contents of the system log file configured in Step 7. The output shows that the ge-0/3/1.0 interface went down and generated an SNMP_TRAP_LINK_DOWN event.

```
bsmith@R1> show log syslog-event-daemon-warning
Oct 10 18:00:57  R1 mib2d[1371]: SNMP_TRAP_LINK_DOWN: ifIndex 531, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
```

### *Verifying the Commit*

**Purpose** Verify that the event policy commit operation was successful by reviewing the commit log and the messages log file.

**Action** Issue the **show system commit** operational mode command to view the commit log. In this example, the log confirms that the configuration was committed through the event policy under the privileges of user bsmith at the given date and time.

```
bsmith@R1> show system commit
0   2011-10-10 18:01:03 PDT by bsmith via junoscript
    updating configuration from event policy update-on-snmp-trap-link-down
1   2011-09-02 14:16:44 PDT by admin via netconf
2   2011-07-08 14:33:46 PDT by root via other
```

Review the **messages** log file. Upon receipt of the SNMP_TRAP_LINK_DOWN event, Junos OS executed the configured event policy action to modify and commit the configuration. The commit operation occurred under the privileges of user bsmith.

```
bsmith@R1> show log messages | last 20
...
Oct 10 18:00:57  R1 mib2d[1371]: SNMP_TRAP_LINK_DOWN: ifIndex 531, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
Oct 10 18:00:59  R1 file[17575]: UI_COMMIT: User 'bsmith' requested 'commit'
operation (comment: updating configuration from event policy
update-on-snmp-trap-link-down)
Oct 10 18:01:03  R1 eventd: EVENTD_CONFIG_CHANGE_SUCCESS: Configuration change
successful: while executing policy update-on-snmp-trap-link-down with user bsmith
 privileges
```

NOTE: If you configure a different log file, review the file specific to your configuration.

**Meaning** The output from the **show system commit** operational mode command and the **messages** log file verify that the commit operation, which was made through the event policy under the privileges of the user bsmith, was successful. The **show system commit** output and **messages** log file reference the commit comment specified in the **log** statement at the **[edit event-options policy update-on-snmp-trap-link-down then change-configuration commit-options]** hierarchy level.

### *Verifying the Configuration Changes*

**Purpose** Verify the configuration changes by reviewing the **[edit routing-options static]** hierarchy level of the configuration after disabling the ge-0/3/1 interface.

**Action** Issue the following operational mode command:

```
bsmith@R1> show configuration routing-options static
...
route 10.1.10.0/24 next-hop 10.1.3.1;
...
```

**Meaning** The configured next hop has been modified by the event policy to the new IP address 10.1.3.1, which has its route through the exit interface ge-0/2/1.

### Troubleshooting



***Troubleshooting Commit Errors***

**Problem** The triggered event policy does not make the specified configuration changes, and the logs verify that the commit was unsuccessful.

```
bsmith@R1> show log messages | last 20
...
Oct 10 17:48:59  R1 mib2d[1371]: SNMP_TRAP_LINK_DOWN: ifIndex 531, ifAdminStatus
 down(2), ifOperStatus down(2), ifName ge-0/3/1.0
Oct 10 17:49:01  R1 file[17142]: UI_LOAD_EVENT: User 'bsmith' is performing a
'rollback'
Oct 10 17:49:01  R1 eventd: EVENTD_CONFIG_CHANGE_FAILED: Configuration change
failed: rpc to management daemon failed while executing policy
update-on-snmp-trap-link-down with user bsmith privileges
```

A failed commit might occur if the configuration is locked or if the configuration mode commands have the incorrect syntax or order.

**Solution** Check the configuration mode commands at the **[edit event-options policy update-on-snmp-trap-link-down then change-configuration commands]** hierarchy level, and verify that the syntax and the order of execution are correct.

Additionally, increase the retry count and interval options so that if the configuration is locked, the event policy attempts the configuration changes a specified number of times after the first failed instance.

## Example: Changing the Interface Configuration in Response to an Event

It might be necessary to modify the configuration in response to a particular event. Starting with Junos OS Release 12.1, you can configure an event policy to make and commit configuration changes when the event policy is triggered by one or more specific events.

This example uses a real-time performance monitoring (RPM) probe to generate PING_TEST_FAILED events for a given interface. Upon receipt of the first instance of two PING_TEST_FAILED events within a 5-minute period from the configured RPM probe, the event policy executes a change configuration event policy action that modifies the configuration to administratively disable the specified interface. This type of action might be necessary if you have an unstable, flapping interface that is consistently affecting network performance.

## Requirements

- Routing, switching, or security device running Junos OS Release 12.1 or later.

## Overview

This example creates an event policy named disable-interface-on-ping-failure. The event policy is configured so that the eventd process listens for PING_TEST_FAILED events generated by a specific RPM probe and associated with the ge-0/3/1 interface . If two PING_TEST_FAILED events occur for the given interface within a 5-minute interval, the event policy executes a change configuration action. The event policy configuration commands administratively disable the interface.

To test the event policy, the example configures an RPM probe that pings the IP address associated with the ge-0/3/1 interface every 60 seconds. In this example, the ge-0/3/1.0 inteface is configured with the IPv4 address 10.1.4.1/26. If the ping fails, the RPM probe generates a PING_TEST_FAILED event. Because multiple RPM tests could be running simultaneously, the event policy matches the owner-name and test-name attributes of the received PING_TEST_FAILED events to the RPM probe owner name and test name. When the RPM probe generates two PING_TEST_FAILED events, it triggers the event policy, which disables the interface.

This policy also demonstrates how to restrict the execution of the same configuration change multiple times because of occurrences of the same event or correlated events. In this example, the **trigger on 1** statement specifies that only the first occurrence of two correlated PING_TEST_FAILED events triggers the configuration change. The PING_TEST_FAILED events must occur within a 5-minute interval (300 seconds) to trigger the event policy.

## Configuration

***Configuring the RPM Probe***

**CLI Quick Configuration**

To quickly configure this section of the example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set services rpm probe icmp-ping-probe test ping-probe-test
set services rpm probe icmp-ping-probe test ping-probe-test probe-type icmp-ping
    test-interval 60
set services rpm probe icmp-ping-probe test ping-probe-test probe-type icmp-ping target
    address 10.1.4.1
set system syslog file syslog-event-daemon-info daemon info
```

**Step-by-Step Procedure**

To configure the RPM probe, which creates the PING_TEST_FAILED events for this example:

1. Create an RPM probe named ping-probe-test at the **[edit services rpm]** hierarchy level to ping the ge-0/3/1 interface.

```
[edit services rpm]
bsmith@R1# set probe icmp-ping-probe test ping-probe-test
```

2. Configure the RPM probe to send ICMP echo requests to the ge-0/3/1 interface at IP address 10.1.4.1, and set **test-interval** to 60 to issue the test every 60 seconds.

   ```
   [edit services rpm probe icmp-ping-probe test ping-probe-test]
   bsmith@R1# set probe-type icmp-ping test-interval 60 target address 10.1.4.1
   ```

3. Configure a new log file at the **[edit system syslog]** hierarchy level to record syslog events of facility **daemon** and severity **info**.

   This captures the events sent during the probe tests.

   ```
   [edit system syslog]
   bsmith@R1# set file syslog-event-daemon-info daemon info
   ```

4. Commit the configuration.

   ```
   bsmith@R1# commit
   ```

**Results**

```
[edit]
services {
  rpm {
    probe icmp-ping-probe {
      test ping-probe-test {
        probe-type icmp-ping;
        target address 10.1.4.1;
        test-interval 60;
      }
    }
  }
}
system {
  syslog {
    file syslog-event-daemon-info {
      daemon info;
    }
  }
}
```

*Configuring the Event Policy*

**CLI Quick Configuration**

To quickly configure this section of the example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level:

```
set event-options policy disable-interface-on-ping-failure events ping_test_failed
set event-options policy disable-interface-on-ping-failure within 300 trigger on
set event-options policy disable-interface-on-ping-failure within 300 trigger 1
set event-options policy disable-interface-on-ping-failure attributes-match
  ping_test_failed.test-owner matches icmp-ping-probe
set event-options policy disable-interface-on-ping-failure attributes-match
  ping_test_failed.test-name matches ping-probe-test
set event-options policy disable-interface-on-ping-failure then change-configuration
  retry count 5
set event-options policy disable-interface-on-ping-failure then change-configuration
  retry interval 4
```

```
set event-options policy disable-interface-on-ping-failure then change-configuration
  commands "set interfaces ge-0/3/1 disable"
set event-options policy disable-interface-on-ping-failure then change-configuration
  user-name bsmith
set event-options policy disable-interface-on-ping-failure then change-configuration
  commit-options log "updating configuration from event policy
  disable-interface-on-ping-failure"
```

**Step-by-Step Procedure**

1. Create and name the event-policy.

   ```
   [edit]
   bsmith@R1# edit event-options policy disable-interface-on-ping-failure
   ```

2. Configure the event policy to match on the PING_TEST_FAILED event if it occurs twice within 5 minutes (300 seconds).

   The **trigger on 1** statement specifies that only the first set of correlated PING_TEST_FAILED events triggers this policy.

   The **attributes-match** statement is set so that the policy triggers only on the PING_TEST_FAILED events generated by the associated RPM probe.

   ```
   [edit event-options policy disable-interface-on-ping-failure]
   bsmith@R1# set events ping_test_failed
   bsmith@R1# set within 300 trigger on 1
   bsmith@R1# set attributes-match ping_test_failed.test-owner matches
     icmp-ping-probe
   bsmith@R1# set attributes-match ping_test_failed.test-name matches
     ping-probe-test
   ```

3. Specify the configuration mode commands that are executed if the event policy triggers.

   Configure each command on a single line, enclose the command string in quotes, and specify the complete statement path.

   ```
   [edit event-options policy disable-interface-on-ping-failure then
     change-configuration]
   bsmith@R1# set commands "set interfaces ge-0/3/1 disable"
   ```

4. (Optional) Configure the retry count and retry interval.

   In this example, **count** is set to 5, and the **interval** is 4 seconds.

   ```
   [edit event-options policy disable-interface-on-ping-failure then
     change-configuration]
   bsmith@R1# set retry count 5 interval 4
   ```

5. Configure the commit options.

   Configure the **log** option with a comment describing the configuration changes. The comment is added to the commit logs after a successful commit operation is made through the associated event policy.

   ```
   [edit event-options policy disable-interface-on-ping-failure then
     change-configuration]
   bsmith@R1# set commit-options log "updating configuration from event policy
     disable-interface-on-ping-failure"
   ```

If you have dual Routing Engines, configure the **synchronize** option to commit the configuration on both Routing Engines. Include the **force** option to force the commit on the other Routing Engine, ignoring any warnings. This example does not configure the **synchronize** and **force** options.

6. (Optional) Configure the username under whose privileges the configuration changes and commit are made.

   If you do not specify a username, the action is executed as user **root**.

   ```
   [edit event-options policy disable-interface-on-ping-failure then
     change-configuration]
   bsmith@R1# set user-name bsmith
   ```

7. Commit the configuration.

   ```
   bsmith@R1# commit
   ```

8. Review the output of the **show interfaces ge-0/3/1** operational mode command before the configuration change takes place.

   The interface should be enabled.

   ```
   bsmith@R1> show interfaces ge-0/3/1
   Physical interface: ge-0/3/1, Enabled, Physical link is Up
     Interface index: 142, SNMP ifIndex: 531
   ...
   ```

*Results*

```
[edit event-options]
policy disable-interface-on-ping-failure {
  events ping_test_failed;
  within 300 {
    trigger on 1;
  }
  attributes-match {
    ping_test_failed.test-owner matches icmp-ping-probe;
    ping_test_failed.test-name matches ping-probe-test;
  }
  then {
    change-configuration {
      retry count 5 interval 4;
      commands {
        "set interfaces ge-0/3/1 disable";
      }
      user-name bsmith;
      commit-options log "updating configuration from event policy
        disable-interface-on-ping-failure";
    }
  }
}
```

## Verification

Confirm that the configuration is working properly.



### *Verifying the Events*

**Purpose** To manually test the event policy, take the ge-0/3/1 interface offline until two PING_TEST_FAILED events are generated.

**Action** Review the configured syslog file. Verify that when the RPM probe ping tests fail, the probe generates a PING_TEST_FAILED event.

```
bsmith@R1> show log syslog-event-daemon-info
Oct  7 15:48:54  R1 rmopd[1345]: PING_TEST_COMPLETED: pingCtlOwnerIndex =
icmp-ping-probe, pingCtlTestName = ping-probe-test
Oct  7 15:49:54  R1 rmopd[1345]: PING_TEST_COMPLETED: pingCtlOwnerIndex =
icmp-ping-probe, pingCtlTestName = ping-probe-test
...
Oct  7 15:52:54  R1 rmopd[1345]: RMOPD_ICMP_SENDMSG_FAILURE: sendmsg(ICMP): No
route to host
Oct  7 15:52:54  R1 rmopd[1345]: PING_PROBE_FAILED: pingCtlOwnerIndex =
icmp-ping-probe, pingCtlTestName = ping-probe-test
Oct  7 15:52:54  R1 rmopd[1345]: PING_TEST_FAILED: pingCtlOwnerIndex =
icmp-ping-probe, pingCtlTestName = ping-probe-test
Oct  7 15:52:57  R1 rmopd[1345]: PING_TEST_FAILED: pingCtlOwnerIndex =
icmp-ping-probe, pingCtlTestName = ping-probe-test
```

### *Verifying the Commit*

**Purpose** Verify that the event policy commit operation was successful by reviewing the commit log and the messages log file.

**Action** Issue the **show system commit** operational mode command to view the commit log. In this example, the log confirms that the configuration was committed through the event policy under the privileges of user bsmith at the given date and time.

```
bsmith@R1> show system commit
0   2011-10-07 15:52:58 PDT by bsmith via junoscript
    updating configuration from event policy disable-interface-on-ping-failure
1   2011-09-02 14:16:44 PDT by admin via netconf
2   2011-07-08 14:33:46 PDT by root via other
```

Review the messages log file. Upon receipt of the PING_TEST_FAILED event, Junos OS executed the configured event policy action to modify and commit the configuration. The commit operation occurred under the privileges of user bsmith.

```
bsmith@R1> show log messages | last 20
Oct  7 15:52:54  R1 rmopd[1345]: RMOPD_ICMP_SENDMSG_FAILURE: sendmsg(ICMP): No
route to host
Oct  7 15:52:55  R1 file[9972]: UI_COMMIT: User 'bsmith' requested 'commit'
operation (comment: updating configuration from event policy
disable-interface-on-ping-failure)
Oct  7 15:52:59  R1 eventd: EVENTD_CONFIG_CHANGE_SUCCESS: Configuration change
successful: while executing policy disable-interface-on-ping-failure with user
bsmith privileges
```

**Meaning** The output from the **show system commit** operational mode command and the **messages** log file verify that the commit operation, which was made through the event policy under the privileges of the user bsmith, was successful. The **show system commit** output and **messages** log file reference the commit comment specified in the **log** statement at the **[edit event-options policy disable-interface-on-ping-failure then change-configuration commit-options]** hierarchy level.

*Verifying the Configuration Changes*

**Purpose** Verify the configuration changes by reviewing the **[edit interfaces ge-0/3/1]** hierarchy level of the configuration.

**Action**

```
bsmith@R1> show configuration interfaces ge-0/3/1
disable;
unit 0 {
  family inet {
    address 10.1.4.1/26;
  }
}
```

**Meaning** The ge-0/3/1 configuration hierarchy was modified through the event policy to add the **disable** statement.

*Verifying the Status of the Interface*

**Purpose** Review the output of the **show interfaces ge-0/3/1** operational mode command after the configuration change takes place.

**Action** Issue the **show interfaces ge-0/3/1** operational mode command. After the event policy configuration change action disables the interface, the output changes from "Enabled" to "Administratively down".

```
bsmith@R1> show interfaces ge-0/3/1
Physical interface: ge-0/3/1, Administratively down, Physical link is Down
  Interface index: 142, SNMP ifIndex: 531
```

**Related Documentation**

- change-configuration on page 627
- commands (Event Policy Change Configuration) on page 628
- commit-options on page 630
- retry (Event Policy) on page 646
- user-name on page 658

CHAPTER 30

# Event Policy Examples

This chapter includes the following examples:



## Example: Assigning a Transfer Delay to an Event Policy Action

This section discusses three examples.

**Example 1** Configure two event policies, **policy1** and **policy2**. The **policy1** event policy has a 5-second transfer-delay when uploading the **process.core** file to the **some-dest** destination. The **policy2** event policy has no transfer delay when uploading the **process.core** file to the same destination.

```
[edit event-options]
policy policy1 {
   events e1;
   then {
      upload filename process.core destination some-dest {
         transfer-delay 5;
      }
   }
}
policy policy2 {
   events e2;
```

```
    then {
      upload filename process.core destination some-dest;
    }
  }
  destinations {
    some-dest {
      archive-sites {
        "scp://robot@my.little.com/foo/moo" password "password";
        "scp://robot@my.big.com/foo/moo" password "password";
      }
    }
  }
```

**Example 2** The **policy1** event policy has a 7-second (5 seconds + 2 seconds) transfer delay when uploading the **process.core** file to the destination. The **policy2** event policy has a 2-second transfer delay when uploading the **process.core** file to the destination.

```
[edit event-options]
policy policy1 {
  events e1;
  then {
    upload filename process.core destination some-dest {
      transfer-delay 5;
    }
  }
}
policy policy2 {
  events e2;
  then {
    upload filename process.core destination some-dest;
  }
}
destinations {
  some-dest {
    transfer-delay 2;
    archive-sites {
      "scp://robot@my.little.com/foo/moo" password "password";
      "scp://robot@my.big.com/foo/moo" password "password";
    }
  }
}
```

**Example 3** The **policy1** event-policy is executed with **user1** privileges and uploads the **process.core** file after a transfer delay of 7 seconds (5 seconds + 2 seconds). The **policy2** event policy is executed with **root** privileges and uploads the **process.core** file after a transfer delay of 6 seconds (4 seconds + 2 seconds).

```
[edit event-options]
policy policy1 {
  events e1;
  then {
    upload filename process.core destination some-dest {
      transfer-delay 5;
      user-name user1;
    }
  }
```

```
}
policy policy2 {
  events e2;
  then {
    upload filename process.core destination some-dest {
      transfer-delay 4;
    }
  }
}
destinations {
  some-dest {
    transfer-delay 2;
    archive-sites {
      "scp://robot@my.little.com/foo/moo" password "password";
      "scp://robot@my.big.com/foo/moo" password "password";
    }
  }
}
```

## Example: Associating an Optional User with an Event Policy Action

Configure two event policies, **policy1** and **policy2**.

In **policy1**, associate user **user1** with the **execute-commands** action. The **execute-commands** action is executed with **user1** privileges.

In **policy2**, do not explicitly associate a user with the **event-script** action. The **event-script** action is executed with **root** privileges.

```
[edit system]
login {
  user user1 {
    class operator;
  }
}
[edit event-options]
policy p1 {
  events e1;
  then {
    execute-commands {
      commands {
        "show version";
      }
      user-name user1;
      output-filename command-output.txt;
      destination some-dest;
    }
  }
}
policy p2 {
  events e2;
  then {
    event-script script.xsl {
      output-filename event-script-output.txt;
      destination some-dest;
```

```
    }
  }
}
```

## Example: Controlling Event Policy Using a Regular Expression

The following policy is executed only if the **interface-name** attribute in both traps (**SNMP_TRAP_LINK_DOWN** and **SNMP_TRAP_LINK_UP**) match each other and the **interface-name** attribute in the **SNMP_TRAP_LINK_DOWN** trap starts with letter *t*. This means the policy is executed only for T1 (**t1-**) and T3 (**t3-**) interfaces. The policy is not executed when the **eventd** process receives traps from other interfaces.

NOTE: In system log files, the message tags appear in all uppercase letters. In the command-line interface (CLI), the message tags appear in all lowercase letters.

```
[edit event-options]
policy pol6 {
  events snmp_trap_link_down;
  within 120 events snmp_trap_link_up;
  attributes-match {
    snmp_trap_link_up.interface-name equals snmp_trap_link_down.interface-name;
    snmp_trap_link_down.interface-name matches "^t";
  }
  then {
    execute-commands {
      commands {
        "show interfaces {$$.interface-name}";
        "show configuration interfaces {$$.interface-name}";
      }
      output-filename config.txt;
      destination bsd2;
      output-format text;
    }
  }
}
```

## Example: Correlating Events Based on Event Attributes

In the following policy, the two events are correlated only if two of their parameter values match. Matching on attributes of both events ensures that the two events are related. In this case, the interface addresses must match and the physical interface (ifd) names must match.

The **RPD_KRT_IFDCHANGE** error occurs when the routing protocol process (rpd) sends a request to the kernel to change the state of an interface and the request fails. The **RPD_RDISC_NOMULTI** error occurs when an interface is configured for router discovery but the interface does not support IP multicast operations as required.

In this example, **RPD_RDISC_NOMULTI.interface-name** might be so-0/0/0.0, and **RPD_KRT_IFDCHANGE.ifd-index** might be so-0/0/0.

```
[edit event-options]
policy 1 {
   events rpd_rdisc_nomulti;
   within 500 events rpd_krt_ifdchange;
   attributes-match {
      rpd_rdisc_nomulti.interface-address equals rpd_krt_ifdchange.address;
      rpd_rdisc_nomulti.interface-name starts-with rpd_krt_ifdchange.ifd-index;
   }
   then {
      ... actions ...
   }
}
```

## Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval

In the following policy, a set of commands is issued and the output is logged and saved to a given location. The policy is executed if ***event3***, ***event4***, or ***event5*** occurs within 60 seconds after ***event1*** or ***event2*** occurs. The pseudocode for the policy is as follows:

```
if this event is (event3 or event4 or event5)
  and
  (event1 or event2 has been received within the last 60 seconds)
then {
  run a set of commands;
  log the output of these commands to a location;
}
```

Specify two archive sites in the configuration. The device attempts to transfer to the first archive site in the list, moving to the next site only if the transfer fails.

```
[edit event-options]
policy 1 {
   events [ event3 event4 event5 ];
   within 60 events [ event1 event2 ];
   then {
      execute-commands {
         commands {
            "command";
         }
         output-filename my_cmd_out;
         destination policy-1-command-dest;
      }
   }
}
destinations {
   policy-1-command-dest {
      archive-sites {
         scp://robot@my.big.com/a/b;
         scp://robot@my.little.com/a/b;
      }
   }
}
```

## Example: Generating an Internal Event

The following two examples generate an internal event. In the first example, the configuration generates an internal event every hour. In the second example, the configuration generates an event every night at midnight.

In the following example, the internal event called **EVERY-ONE-HOUR** is generated every hour (3600 seconds). If 3601 seconds pass and the event has not been generated, certain actions are taken.

```
[edit event-options]
generate-event every-one-hour time-interval 3600;
policy check-heartbeat {
   events every-one-hour;
   within 3601 not events every-one-hour;
   then {
      ... actions ...
   }
}
```

In the following example, the internal event called **IT-IS-MIDNIGHT** is generated at 12:00 AM every night (00:00:00). When the eventd process receives the **IT-IS-MIDNIGHT** event, certain actions are taken.

```
[edit event-options]
generate-event it-is-midnight time-of-day 00:00:00;
policy midnight-chores {
   events it-is-midnight;
   then {
      ... actions ...
   }
}
```

## Example: Ignoring Events Based on Receipt of Other Events

In the following policy, if any of ***event1***, ***event2***, or ***event3*** has occurred, and either ***event4*** or ***event5*** has occurred within the last 600 seconds, and ***event6*** has not occurred within the last 800 seconds, then the event that triggered the policy (***event1***, ***event2***, or ***event3***) is ignored, meaning system log messages are not created.

```
[edit event-options]
policy 1 {
   events [ event1 event2 event3 ];
   within 600 events [ event4 event5 ];
   within 800 not events event6;
   then {
      ignore;
   }
}
```

Sometimes events are generated repeatedly within a short period of time. In this case, it is redundant to execute a policy multiple times, once for each instance of the event. Event dampening allows you to slow down the execution of policies by ignoring instances of an event that occur within a specified time after another instance of the same event.

In the following example, an action is taken only if the eventd process has not received another instance of the event within the past 60 seconds. If an instance of the event has been received within the last 5 seconds, the policy is not executed and a system log message for the event is not created again.

```
[edit event-options]
policy dampen-policy {
   events event1;
   within 60 events event1;
   then {
      ignore;
   }
}
policy policy {
   events event1;
   then {
      ... actions ...
   }
}
```

## Example: Representing the Correlating Event in an Event Policy

```
[edit event-options]
policy p1 {
   events [ e1 e2 e3 ];
   within 60 events [ e4 e5 e6 ];
   then {
      execute-commands {
         commands {
            "show interfaces {$$.interface-name}";
            "show interfaces {$e4.interface-name}";
            "show interfaces {$*.interface-name}";
         }
         output-filename command-output.txt;
         destination some-dest;
      }
   }
}
```

In the **show interfaces {$$.interface-name}** command, the value of the **interface-name** attribute of event **e1**, **e2**, or **e3** is substituted for the **{$$.interface-name}** variable.

In the **show interfaces {$e4.interface-name}** command, the value of the **interface-name** attribute of the most recent **e4** event is substituted for the **{$e4.interface-name}** variable.

In the **show interfaces {$*.interface-name}** command, the value of the **interface-name** attribute of the most recent **e4**, **e5**, or **e6** event is substituted for the **{$*.interface-name}** variable. If one of **e4**, **e5**, or **e6** occurs within 60 seconds of **e1**, **e2**, or **e3**, the value of the **interface-name** attribute for that correlating event (**e4**, **e5**, or **e6**) is substituted for the **{$*.interface-name}** variable. If the correlating event does not have an **interface-name**

attribute, the software does not execute the **show interfaces {$*.interface-name}** command.

If both **e4** and **e5** occur within 60 seconds of **e1**, then the value of the **interface-name** attribute for **e4** is substituted for the **{$*.interface-name}** variable. This is because the event process (eventd) searches for correlating events in sequential order as configured in the **within** statement. In this case, the order is **e4** > **e5** > **e6**.

## Example: Retrying the File Upload Action

This section discusses two examples.

**Example 1** Configure a policy that retries the file upload operation two times with a time interval of 5 seconds between retries:

```
event-options {
  policy p1 {
    events e1;
    then {
      execute-commands {
        commands {
          command1;
        }
        output-filename command-output.txt;
        destination some-dest {
          retry-count 2 retry-interval 5;
        }
      }
    }
  }
}
```

**Example 2** Configure a transfer delay of 10 seconds and retry the file upload operation two times with a time interval of 5 seconds between retries:

```
event-options {
  policy p2 {
    events e1;
    then {
      execute-commands {
        commands {
          command1;
        }
        output-filename command-output.txt;
        destination some-dest {
          retry-count 2 retry-interval 5;
          transfer-delay 10;
        }
      }
    }
  }
}
```

The transfer delay is in operation for the first upload attempt only. The policy uploads the **command-output.txt** file after a 10-second transfer delay. If the event process (eventd)

detects failure of the upload operation, eventd retries the upload operation after 5 seconds. The failure detection time can be in the range from 60 to 90 seconds, depending on the transmission protocol, such as FTP.

The following sequence describes the file upload operation with two failed retransmissions:

1. Policy triggers upload operation.
2. Transmission delay of 10 seconds.
3. Policy tries to upload the output file.
4. Policy detects transmission failure.
5. Retry interval of 5 seconds.
6. Policy tries to upload the output file.
7. Policy detects transmission failure.
8. Retry interval of 5 seconds.
9. Policy tries to upload the output file.
10. Policy detects transmission failure.
11. Policy declares the failure of the file upload operation.

## Example: Triggering a Policy Based on Event Count

This section discusses two examples.

NOTE: The RADIUS_LOGIN_FAIL, TELNET_LOGIN_FAIL, and SSH_LOGIN_FAIL events are not actual Junos OS events. They are illustrative for these examples.

**Example 1** Configure an event policy called **login**. The **login** policy is executed if five login failure events (**RADIUS_LOGIN_FAIL**, **TELNET_LOGIN_FAIL**, or **SSH_LOGIN_FAIL**) are generated within 120 seconds. Take action by executing the **login-fail.xsl** event script, which disables the user account.

```
[edit event-options]
policy login {
   events [ RADIUS_LOGIN_FAIL TELNET_LOGIN_FAIL SSH_LOGIN_FAIL ];
   within 120 {
      trigger after 4;
   }
   then {
      event-script login-fail.xsl {
         destination some-dest;
      }
   }
}
```

Table 41 on page 620 shows how events add to the count.

Table 41: Event Count Triggers Policy

| Event Number | Event | Time | Count | Order |
|---|---|---|---|---|
| 1 | RADIUS_LOGIN_FAIL | 00:00:00 | 1 | [1] |
| 2 | TELNET_LOGIN_FAIL | 00:00:20 | 2 | [1 2] |
| 3 | RADIUS_LOGIN_FAIL | 00:02:05 | 2 | [2 3] |
| 4 | SSH_LOGIN_FAIL | 00:02:40 | 2 | [3 4] |
| 5 | TELNET_LOGIN_FAIL | 00:02:55 | 3 | [3 4 5] |
| 6 | TELNET_LOGIN_FAIL | 00:03:01 | 4 | [3 4 5 6] |
| 7 | RADIUS_LOGIN_FAIL | 00:03:55 | 5 | [3 4 5 6 7] |

The columns in Table 41 on page 620 mean the following:

- Event number—Event sequence number.
- Event—Policy login events received by the event process (eventd).
- Time—Time (in ***hh:mm:ss*** format) when eventd receives the event.
- Count—The number of events received by eventd within the last 120 seconds.
- Order—Order of events as received by eventd within the last 120 seconds.

At time 00:03:55, the value of count is more than 4; therefore, the **login** policy executes the **login-fail.xsl** script.

**Example 2** Configure an event policy called **login**. The **login** policy is executed if five login failure events (**RADIUS_LOGIN_FAIL**, **TELNET_LOGIN_FAIL**, or **SSH_LOGIN_FAIL**) are generated within 120 seconds from username **roger**. Take action by executing the **login-fail.xsl** event script, which disables the **roger** user account.

```
[edit event-options]
policy p2 {
   events [ RADIUS_LOGIN_FAIL TELNET_LOGIN_FAIL SSH_LOGIN_FAIL ];
   within 120 {
      trigger after 4;
   }
   attributes-match {
      RADIUS_LOGIN_FAIL.username matches roger;
      TELNET_LOGIN_FAIL.username matches roger;
   }
   then {
      event-script login-fail.xsl {
         destination some-dest;
      }
   }
```

```
}
```

## Example: Using Nonstandard System Log Messages to Trigger an Event Policy

Reference a **KERNEL** system log message in an event policy. The **raise-trap** action in the **then** statement is executed only if a **KERNEL** event containing a message that matches "exited on signal 11" occurs.

```
[edit event-options]
policy kernel-policy {
   events KERNEL;
   attributes-match {
      KERNEL.message matches "exited on signal 11";
   }
   then {
      raise-trap;
   }
}
```

CHAPTER 31

# Summary of Event Policy Configuration Statements

This chapter describes each configuration statement for event policies. The statements are organized alphabetically.

## archive-sites

**Syntax**

```
archive-sites {
  url <password password>;
}
```

**Hierarchy Level** [edit event-options destinations *destination-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Specify an archive site to which files are transferred. If you specify more than one archive site, the device attempts to transfer to the first archive site in the list, moving to the next site only if the transfer fails.

**Options** *url*—The archive destination specified as a file URI, an active or passive FTP URI, or a Secure Copy (SCP) URI. Local device directories are also supported (for example, **/var/tmp/**).

file:<//host>/path

ftp://username@host:<port>url-path

pasvftp://username@host:<port>url-path

scp://username@host:<port>url-path

<path>/<filename>

**password** ***password***—A plain-text password required for logging into the archive site.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Example: Defining Destinations for File Archiving by Event Policies on page 563
- destinations on page 632

## arguments (Event Options)

**Syntax**

```
arguments {
  argument-name argument-value;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then event-script *filename*]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define command-line arguments for an event script that is invoked from an event policy.

**Options** ***argument-name***—Name of the argument.

***argument-value***—Value of the argument.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Executing Event Scripts in an Event Policy on page 578
- event-script (Event Policy) on page 636
- policy (Event Policy) on page 643

## attributes-match

**Syntax**

```
attributes-match {
  event1.attribute-name equals event2.attribute-name;
  event.attribute-name matches regular-expression;
  event1.attribute-name starts-with event2.attribute-name;
}
```

**Hierarchy Level** [edit event-options policy *policy-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Execute the policy only if the attributes of two events are correlated or if the attribute of one event matches a regular expression.

If the **attributes-match** statement includes the **equals** or **starts-with** options, or if it includes a **matches** option that includes a clause for an event that is not specified at the **[edit event-options policy *policy-name* events]** hierarchy level, you must include one or more **within** statements in the same policy configuration.

Starting with Junos OS Release 11.1, you can include event policy variables within the statement to differentiate between a trigger event attribute and a correlated event attribute. You can use variables of the following forms:

- **{$$.*attribute-name*}**—The double dollar sign ($$) notation represents the event that is triggering a policy. When combined with an attribute name, the variable is replaced by the value of the attribute of the triggering event.
- **{$*event.attribute-name*}**—The dollar sign with the event name ($*event*) notation represents the most recent event that matches the specified event. The variable is replaced by the value of the attribute of the most recent event that matches ***event***.

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**

- Using Correlated Events to Trigger an Event Policy on page 555
- Using Regular Expressions to Refine the Set of Events That Trigger a Policy on page 559
- equals (Event Policy) on page 633
- matches on page 640
- starts-with on page 649
- within on page 659

## change-configuration

**Syntax**

```
change-configuration {
  commands {
    "command";
  }
  commit-options {
    check <synchronize>;
    force;
    log "comment-string";
    synchronize;
  }
  retry count number interval seconds;
  user-name username;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** When the associated event policy is invoked, update the candidate configuration using Junos OS configuration mode commands, and commit the changes.

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**

- Configuring an Event Policy to Change the Configuration on page 595
- commands (Event Policy Change Configuration) on page 628
- commit-options on page 630
- retry (Event Policy) on page 646
- user-name on page 658

## commands (Event Policy Change Configuration)

**Syntax**

```
commands {
  "command";
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then change-configuration]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Specify the configuration mode commands to be issued on receipt of an event. Within an event policy, on receipt of the specified event or events, the event process (eventd) invokes the configured commands to update the candidate configuration, which is then committed, provided that no commit errors occur. The eventd process executes the configuration commands in the order in which they appear in the event policy configuration.

**Options** *command*—Configuration mode command to be executed. Enclose each command in quotation marks (**" "**), and specify the complete statement path to the element, identifier, or value as you do in configuration mode when issuing commands at the **[edit]** hierarchy level.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring an Event Policy to Change the Configuration on page 595
- change-configuration on page 627
- commit-options on page 630
- retry (Event Policy) on page 646
- user-name on page 658

## commands (Event Policy Execute Commands)

**Syntax**

```
commands {
  "command";
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then execute-commands]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Specify an operational mode command to be issued on receipt of an event.

**Options** ***command***—Command to be issued. Enclose each command in quotation marks (“ ”). The event process (eventd) issues the commands in the order in which they appear in the configuration.

You can include variables in commands. The eventd process replaces each variable with values contained in the event that triggers the policy. You can use command variables of the following forms:

- **{$$.*attribute-name*}**—The double dollar sign (**$$**) notation represents the event that is triggering a policy. When combined with an attribute name, the command variable is replaced by the value of the attribute name of the triggering event.
- **{$*event*.*attribute-name*}**—The dollar sign with the event name (**$*event***) notation represents the most recent event that matches the specified event. The variable is replaced by the value of the attribute name of the most recent event that matches ***event***.
- **{$*.*attribute-name*}**—The dollar sign with the asterisk (**$***) notation represents the most recent event that matches any of the correlating events. The variable is replaced by the value of the attribute name of the most recent event that matches any of the events specified in the policy configuration.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring an Event Policy to Execute Operational Mode Commands on page 574
- Representing the Correlating Event in an Event Policy on page 558

## commit-options

**Syntax**

```
commit-options {
  check <synchronize>;
  force;
  log "comment-string";
  synchronize;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then change-configuration]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Customize the commit options for configuration updates made through an event policy. The **check** statement and the other **commit-options** statements are mutually exclusive.

**Options**

**check <synchronize>**—Verify that the candidate configuration is syntactically correct, but do not commit the changes. On dual control plane systems, when the **check synchronize** statement is configured, the candidate configuration on one control plane is copied to the other control plane, and the system verifies that both candidate configurations are syntactically correct. The **check** statement and the other **commit-options** statements are mutually exclusive.

**force**—Force the commit on the other Routing Engine, ignoring any warnings. By default, the **synchronize** command does not work if the responding Routing Engine has uncommitted configuration changes. However, you can enforce commit synchronization on the Routing Engines by using the **force** option.

**log "*comment-string*"**—Include a comment describing changes to the committed configuration. Enclose the comment in quotation marks and include it on a single line. To view commit comments, issue the **show system commit** operational mode command.

**synchronize**—Synchronize the commit on both Routing Engines. The Routing Engine on which you execute this command copies and loads its candidate configuration to the other Routing Engine. Both Routing Engines perform a syntax check on the candidate configuration file. If no errors are found, the configuration is activated and becomes the current operational configuration on both Routing Engines.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring an Event Policy to Change the Configuration on page 595
- change-configuration on page 627
- commands (Event Policy Change Configuration) on page 628
- retry (Event Policy) on page 646
- user-name on page 658

## destination (Event Policy)

**Syntax**

```
destination destination-name {
    retry-count count retry-interval seconds;
    transfer-delay seconds;
}
```

**Hierarchy Level**

[edit event-options policy *policy-name* then event-script *filename*],
[edit event-options policy *policy-name* then execute-commands]

**Release Information**

Statement introduced in Junos OS Release 7.5.Support extended to the **[edit event-options policy *policy-name* then event-script *filename*]** hierarchy level in Junos OS Release 7.6. Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description**

Assign a location to which to upload command or script output for the specified policy.

**Options**

***destination-name***—Name of a destination defined in the **destinations** statement at the **[edit event-options]** hierarchy level.

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**

- Configuring an Event Policy to Execute Operational Mode Commands on page 574
- Executing Event Scripts in an Event Policy on page 578
- **destinations on page 632**

## destinations

**Syntax**

```
destinations {
  destination-name {
    archive-sites {
      url <password password>;
    }
    transfer-delay seconds;
  }
}
```

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define one or more destinations, each with a unique name and other attributes. You can use the destination as a storage location for command output and for various files, such as system log files and core files.

**Options** ***destination-name***—Name of a destination.

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**

- Example: Defining Destinations for File Archiving by Event Policies on page 563

## equals (Event Policy)

**Syntax** *event1.attribute-name* equals *event2.attribute-name*;

**Hierarchy Level** [edit event-options policy *policy-name* attributes-match]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Execute the policy only if the specified attribute of ***event1*** equals the specified attribute of ***event2***.

**Options** ***event1.attribute-name***—Attribute of one event.

***event2.attribute-name***—Attribute of another event.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555

## event-options

**Syntax**

```
event-options {
    destinations {
        destination-name {
            archive-sites {
                url <password password>;
            }
            transfer-delay seconds;
        }
    }
    event-script {
        file filename {
            checksum (md5 | sha-256 | sha1) hash;
            refresh;
            refresh-from url;
            remote-execution {
                remote-hostname {
                    passphrase user-password;
                    username user-login;
                }
            }
            source url;
        }
        max-datasize
        refresh;
        refresh-from url;
        traceoptions {
            file <filename> <files number> <size size> <world-readable | no-world-readable>;
            flag flag;
            no-remote-trace;
        }
    }
    generate-event event-name {
        time-interval seconds;
        time-of-day hh:mm:ss;
    }
    max-policies
    policy policy-name {
        attributes-match {
            event1.attribute-name equals event2.attribute-name;
            event.attribute-name matches regular-expression;
            event1.attribute-name starts-with event2.attribute-name;
        }
        events [events];
        then {
            change-configuration {
                commands {
                    "command";
                }
                commit-options {
                    check <synchronize>;
                    force;
                    log "comment-string";
```

```
            synchronize;
          }
          retry count number interval seconds;
          user-name username;
        }
        event-script filename {
          arguments {
            argument-name argument-value;
          }
          destination destination-name {
            retry-count number retry-interval seconds;
            transfer-delay seconds;
          }
          output-filename filename;
          output-format (text | xml);
          user-name name;
        }
        execute-commands {
          commands {
            "command";
          }
          destination destination-name {
            retry-count count retry-interval seconds;
            transfer-delay seconds;
          }
          output-filename filename;
          output-format (text | xml);
          user-name username;
        }
        ignore;
        priority-override {
          facility facility-type;
          severity severity-level;
        }
        raise-trap;
        upload filename (filename | committed) destination destination-name {
          retry-count count retry-interval seconds;
          transfer-delay seconds;
          user-name username;
        }
      }
      within seconds {
        events [ events ];
        not events [ events ];
        trigger (after number | on number | until number);
      }
    }
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag;
    }
}
```

**Hierarchy Level** [edit]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure event policies.

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.

## event-script (Event Policy)

**Syntax**

```
event-script filename {
  arguments {
    argument-name argument-value;
  }
  destination destination-name {
    retry-count count retry-interval seconds;
    transfer-delay seconds;
  }
  output-filename filename;
  output-format (text | xml);
  user-name username;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** On receipt of an event, specify operational mode commands to be issued, the format of the command output, and a name and destination for the output file.

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**

- Executing Event Scripts in an Event Policy on page 578

## events

See the following sections:


- events (Associating Events with a Policy) on page 637
- events (Correlating Events with Each Other) on page 637


### events (Associating Events with a Policy)

**Syntax** events [ *events* ];

**Hierarchy Level** [edit event-options policy *policy-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Create a list of events that trigger this policy. If one or more of the listed events occurs, the policy is executed.

**Options** **[ *events* ]**—List of events. Events can be internally generated, or they can be generated by Junos OS processes.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555
- Example: Correlating Events Based on Event Attributes on page 614

### events (Correlating Events with Each Other)

**Syntax** events [ *events* ];

**Hierarchy Level** [edit event-options policy *policy-name* within *seconds*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Create a list of events that must occur within a specified time interval for the policy to be triggered.

**Options** **[ *events* ]**—List of events. Events can be internally generated, or they can be generated by Junos OS processes.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555

## execute-commands

**Syntax**

```
execute-commands {
  commands {
    "command";
  }
  destination destination-name {
    retry-count count retry-interval seconds;
    transfer-delay seconds;
  }
  output-filename filename;
  output-format (text | xml);
  user-name username;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** On receipt of an event, specify operational mode commands to be issued, the format of the command output, and a name and destination for the output file.

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**

- Configuring an Event Policy to Execute Operational Mode Commands on page 574

## facility

**Syntax** facility *facility-type*;

**Hierarchy Level** [edit event-options policy *policy-name* then priority-override]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Within an event policy, override the default facility type of the triggering event so that the event is logged based on the configured facility type.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring the System Log Priority of the Triggering Event in an Event Policy on page 589
- Junos OS System Logging Facilities and Message Severity Levels
- priority-override on page 645
- severity on page 648

## generate-event

**Syntax**

```
generate-event event-name {
  time-interval seconds;
  time-of-day hh:mm:ss;
}
```

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Generate an internal event, based on a time interval or the time of day. You can configure up to 10 internal events.

**Options** ***event-name***—Name of an internally generated event.

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**

- Generating Internal Events to Trigger Event Policies on page 560
- time-interval on page 651
- time-of-day on page 652

## ignore

**Syntax**

```
ignore;
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define a policy that ignores particular events. If one or more of the listed events occur, a system log message for the event is not generated, and no further policies associated with this event are processed. If you include the **ignore** statement in a policy configuration, you cannot configure any other actions in the policy.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring Event Policies to Ignore an Event on page 582

## matches

**Syntax** *event.attribute-name* matches *regular-expression*;

**Hierarchy Level** [edit event-options policy *policy-name* attributes-match]

**Release Information** Statement introduced in Junos OS Release 7.5.

**Description** Execute the policy only if the specified attribute of ***event*** matches a regular expression.

**Options** ***event.attribute-name***—Event attribute to compare to a regular expression.

***regular-expression***—Regular expression to compare.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555
- Using Regular Expressions to Refine the Set of Events That Trigger a Policy on page 559

## max-policies

**Syntax** max-policies *policies*;

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 12.3.

**Description** Maximum number of event policies that can run concurrently on the device.

**Options** ***policies***—Maximum number of event policies that can run concurrently.
**Range:** 0 through 20
**Default:** 15

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- max-datasize on page 450
- Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts on page 227

## not

**Syntax** not events [ *events* ];

**Hierarchy Level** [edit event-options policy *policy-name* within *seconds*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Create a list of events that must not occur within the specified time interval for the policy to be triggered.

**Options** **[ *events* ]**—List of events. Events can be internally generated, or they can be generated by Junos OS processes.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Using Correlated Events to Trigger an Event Policy on page 555

## output-filename

**Syntax** output-filename *filename*;

**Hierarchy Level** [edit event-options policy *policy-name* then event-script *filename*],
[edit event-options policy *policy-name* then execute-commands]

**Release Information** Statement introduced in Junos OS Release 7.5.
Support at the **[edit event-options policy *policy-name* then event-script *filename*]** hierarchy level introduced in Junos OS Release 7.6.

**Description** Assign a filename to which to write command or script output for the specified commands or script. For op scripts, this statement is optional.

**Options** ***filename***—Name of a file in which to write command or script output.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Configuring an Event Policy to Execute Operational Mode Commands on page 574
- Executing Event Scripts in an Event Policy on page 578

## output-format

**Syntax** output-format (text | xml);

**Hierarchy Level** [edit event-options policy *policy-name* then event-script *filename*],
[edit event-options policy *policy-name* then execute-commands]

**Release Information** Statement introduced in Junos OS Release 7.5.
Support at the **[edit event-options policy *policy-name* then event-script *filename*]** hierarchy level introduced in Junos OS Release 8.3.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Specify the format (ASCII text or XML) for the output of the specified commands or script.

**Default** The default output format is XML at the **[edit event-options policy *policy-name* then execute-commands]** hierarchy level and text at the **[edit event-options policy *policy-name* then event-script *filename*]** hierarchy level.

**Options** **text**—Formatted ASCII text.

**xml**—Junos Extensible Markup Language (XML) tags.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring an Event Policy to Execute Operational Mode Commands on page 574
- Executing Event Scripts in an Event Policy on page 578

## policy (Event Policy)

**Syntax**

```
policy policy-name {
  attributes-match {
    event1.attribute-name equals event2.attribute-name;
    event.attribute-name matches regular-expression;
    event1.attribute-name starts-with event2.attribute-name;
  }
  events [ events ];
  then {
    ... the then subhierarchy appears at the end of the [edit event-options policy policy-name]
      hierarchy level ...
  }
  within seconds {
    events [ events ];
    not events [ events ];
    trigger (on | after | until) event-count;
  }

  then {
    change-configuration {
      commands {
        "command";
      }
      commit-options {
        check <synchronize>;
        force;
        log "comment-string";
        synchronize;
      }
      retry count number interval seconds;
      user-name username;
    }
    event-script filename {
      arguments {
        argument-name argument-value;
      }
      destination destination-name {
        retry-count count retry-interval seconds;
        transfer-delay seconds;
      }
      output-filename filename;
      output-format (text | xml);
      user-name username;
    }
    execute-commands {
      commands {
        "command";
      }
      destination destination-name {
        retry-count count retry-interval seconds;
        transfer-delay seconds;
      }
      output-filename filename;
```

```
        output-format (text | xml);
        user-name username;
      }
      ignore;
      priority-override {
        facility facility-type;
        severity severity-level;
      }
      raise-trap;
      upload filename (filename | committed) destination destination-name {
        retry-count count retry-interval seconds;
        transfer-delay seconds;
        user-name username;
      }
    }
}
```

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define an event policy to be processed by the eventd process. If you configure a policy, the **events** and **then** statements are mandatory.

You can configure multiple policies to be processed for an event. The policies are executed in the order in which they appear in the configuration. If you configure more than one policy for an event, and if one of the policies is to ignore the event, no policies that follow the **ignore** statement are executed.

**Default** If you do not configure a policy for an event, the event is recorded in the system log.

**Options** *policy-name*—Name of an event policy.

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.

## priority-override

**Syntax**

```
priority-override {
  facility facility-type;
  severity severity-level;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Within an event policy, override the default system log priority of the triggering event so that the system logs the event with a different facility type, severity level, or both. If you configure multiple event policies to override the priority of the same event, the event is logged based on the priority set by the last executed event policy to change it.

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**

- Configuring the System Log Priority of the Triggering Event in an Event Policy on page 589
- Junos OS System Logging Facilities and Message Severity Levels
- facility on page 638
- severity on page 648

## raise-trap

**Syntax**

```
raise-trap;
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 8.1.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define a policy that raises an SNMP trap in response to an event. If one or more of the listed events occur, the system log message for the event is converted into a trap. This enables an agent to notify a trap-based network management system (NMS) of significant events.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Example: Configuring Event Policies to Raise SNMP Traps on page 584

## retry (Event Policy)

**Syntax** retry count *number* interval *seconds*;

**Hierarchy Level** [edit event-options policy *policy-name* then change-configuration]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Specify the number of times that Junos OS attempts the **change-configuration** event policy action if the initial attempt fails while acquiring a lock on the configuration database. If you include the **retry** statement, you must configure both the **count** and **interval** statements.

**Default** If you do not include the **retry** statement, and the **change-configuration** event policy action fails, the configuration changes specified in the event policy are not implemented or committed.

**Options**

**count *number***—The number of attempts to retry the **change-configuration** event policy action upon failure of the initial attempt.

**Range:** 0 through 10

**Default:** 0

**interval *seconds***—The time interval specified in seconds between retry attempts.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring an Event Policy to Change the Configuration on page 595
- change-configuration on page 627
- commands (Event Policy Change Configuration) on page 628
- commit-options on page 630
- user-name on page 658

## retry-count (Event Policy)

**Syntax** retry-count *number* retry-interval *seconds*;

**Hierarchy Level**
[edit event-options policy *policy-name* then event-script *filename* destination *destination-name*],
[edit event-options policy *policy-name* then execute-commands destination *destination-name*],
[edit event-options policy *policy-name* then upload filename (*filename* | committed) destination *destination-name*]

**Release Information**
Statement introduced in Junos OS Release 8.4.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure an event policy to retry a file upload operation if the first attempt fails.

**Default** If you do not include this statement, the file upload operation is attempted one time only.

**Options**
***number***—Number of retries.

**retry-interval *seconds***—Length of time to wait between retries.

**Required Privilege Level**
maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Configuring an Event Policy to Retry the File Upload Action on page 574

## severity

**Syntax** severity *severity-level*;

**Hierarchy Level** [edit event-options policy *policy-name* then priority-override]

**Release Information** Statement introduced in Junos OS Release 12.1.

**Description** Within an event policy, override the preassigned severity level of a triggering event so that the event is logged based on the configured severity level.

**Options** ***severity-level***—Severity level logged for the triggering event. Table 42 on page 648 lists the possible severity levels.

Table 42: System Log Message Severity Levels

| Severity Level | Description |
|---|---|
| emergency | System panic or other conditions that cause the routing platform to stop functioning |
| alert | Conditions that require immediate correction, such as a corrupted system database |
| critical | Critical conditions, such as hard drive errors |
| error | Error conditions that generally have less serious consequences than errors in the emergency, alert, and critical levels |
| warning | Conditions that warrant monitoring |
| notice | Conditions that are not errors but might warrant special handling |
| info | Events or non-error conditions of interest |

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Configuring the System Log Priority of the Triggering Event in an Event Policy on page 589
- Junos OS System Logging Facilities and Message Severity Levels
- facility on page 638
- priority-override on page 645

## starts-with

**Syntax** *event1.attribute-name* starts-with *event2.attribute-name*;

**Hierarchy Level** [edit event-options policy *policy-name* attributes-match *event1.attribute-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.

**Description** Execute the policy only if the specified attribute of ***event1*** starts with the specified attribute of ***event2***.

**Options** ***event1.attribute-name***—Attribute of one event.

***event2.attribute-name***—Attribute of another event.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555

## then

**Syntax**

```
then {
  change-configuration {
    commands {
      "command";
    }
    commit-options {
      check <synchronize>;
      force;
      log "comment-string";
      synchronize;
    }
    retry count number interval seconds;
    user-name username;
  }
  event-script filename {
    arguments {
      argument-name argument-value;
    }
    destination destination-name {
      retry-count count retry-interval seconds;
      transfer-delay seconds;
    }
    output-filename filename;
    output-format (text | xml);
    user-name username;
  }
  execute-commands {
    commands {
      "command";
    }
    destination destination-name {
      retry-count count retry-interval seconds;
      transfer-delay seconds;
    }
    output-filename filename;
    output-format (text | xml);
    user-name username;
  }
  ignore;
  priority-override {
    facility facility-type;
    severity severity-level;
  }
  raise-trap;
  upload filename (filename | committed) destination destination-name {
    retry-count count retry-interval seconds;
    transfer-delay seconds;
    user-name username;
  }
}
```

**Hierarchy Level** [edit event-options policy *policy-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define actions to take if an event occurs. For each policy, you can configure multiple actions.

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**
- Example: Configuring an Event Policy to Upload Files on page 566
- Configuring an Event Policy to Execute Operational Mode Commands on page 574
- Executing Event Scripts in an Event Policy on page 578
- Configuring Event Policies to Ignore an Event on page 582
- Example: Configuring Event Policies to Raise SNMP Traps on page 584

## time-interval

**Syntax** time-interval *seconds*;

**Hierarchy Level** [edit event-options generate-event *event-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure a frequency at which to generate a particular event.

**Options** *seconds*—Time interval between internally generated events.
**Range:** 60 through 2,592,000 seconds

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Generating Internal Events to Trigger Event Policies on page 560
- generate-event on page 639
- time-of-day on page 652

## time-of-day

**Syntax** time-of-day *hh*:*mm*:*ss*;

**Hierarchy Level** [edit event-options generate-event *event-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure a time of day at which to generate a particular event.

**Options** ***hh*:*mm*:*ss***—Time of day at which to generate an event.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Generating Internal Events to Trigger Event Policies on page 560
- generate-event on page 639
- time-interval on page 651

## traceoptions (Event Options)

**Syntax**

```
traceoptions {
  file <filename> <files number> <match regular-expression> <size size> <world-readable |
    no-world-readable>;
  flag flag;
  no-remote-trace;
}
```

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 7.5.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define tracing operations for event policies.

**Default** If you do not include this statement, no event-policy-specific tracing operations are performed.

**Options**

**file *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 **eventd**. If you include the **file** statement, you must specify a filename. To retain the default, you can specify **eventd** as the filename.

**Default:** /var/log/eventd

**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:** 3 files

**flag *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
- **configuration**—Log reading of configuration at the **[edit event-options]** hierarchy level
- **events**—Log eventd processing
- **database**—Log events involving storage and retrieval in events database
- **server**—Log communication with processes that are generating events
- **timer-events**—Log internally generated events

**match *regular-expression***—(Optional) Refine the output to include lines that contain the regular expression.

**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 the ***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 filename.

**Syntax:** *size* to specify bytes, *size*k to specify KB, *size*m to specify MB, or *size*g to specify GB

**Range:** 10 KB through 1 GB

**Default:** 128 KB

**world-readable**—(Optional) 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 Event Policy Processing on page 586

## transfer-delay

**Syntax** transfer-delay *seconds*;

**Hierarchy Level**
[edit event-options destinations *destination-name*],
[edit event-options policy *policy-name* then event-script *filename* destination *destination-name*],
[edit event-options policy *policy-name* then execute-commands destination *destination-name*],
[edit event-options policy *policy-name* then upload filename (*filename* | committed) destination *destination-name*]

**Release Information**
Statement introduced in Junos OS Release 7.5.
Support at the **[edit event-options policy *policy-name* then ...]** hierarchy levels introduced in Junos OS Release 8.4.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure a delay before transferring files. This allows the files to be completely generated before the upload starts. If you configure a transfer delay at the **[edit event-options destination *destination-name*]** hierarchy level and at one of the **[edit event-options policy *policy-name* then ...]** hierarchy levels, the resulting delay is the sum of the two delays.

**Default** If you do not include this statement, there is no transfer delay.

**Options** ***seconds***—Duration of the delay before files are uploaded.

**Required Privilege Level**
maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**
- Example: Defining Destinations for File Archiving by Event Policies on page 563
- Configuring the Delay Before Files Are Uploaded by an Event Policy on page 572

## trigger

**Syntax** trigger (on | after | until) *event-count*;

**Hierarchy Level** [edit event-options policy *policy-name* within *seconds*]

**Release Information** Statement introduced in Junos OS Release 8.4.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Configure an event policy to be triggered if an event or set of events occurs ***event-count*** times within a specified time period.

**Default** If you do not include this statement, the policy is executed on receipt of the first configured event.

**Options**

**after *event-count***—The policy is executed when the number of matching events received equals ***event-count*** + 1.

**on *event-count***—The policy is executed when the number of matching events received equals ***event-count***.

**until *event-count***—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals ***event-count***.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Triggering an Event Policy Based on Event Count on page 559

## upload

**Syntax**

```
upload filename  (filename | committed) destination destination-name {
    retry-count count retry-interval seconds;
    transfer-delay seconds;
    user-name username;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in Junos OS Release 7.5.
**committed** option for **filename** statement introduced in Junos OS Release 8.1.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** On receipt of an event, upload the committed configuration file to a destination.

**Options**

**destination *destination-name***—Name of the destination for the uploaded file. It must be defined in the **destinations** statement at the **[edit event-options]** hierarchy level.

**filename (*filename* | committed)**—Name of the file to upload. Specify either the word **committed** to upload the most recently committed configuration file, or the filename of another file.

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**

- destinations on page 632
- Example: Configuring an Event Policy to Upload Files on page 566

## user-name

**Syntax** user-name *username*;

**Hierarchy Level** [edit event-options policy *policy-name* then change-configuration],
[edit event-options policy *policy-name* then event-script *filename*],
[edit event-options policy *policy-name* then execute-commands],
[edit event-options policy *policy-name* then upload filename (*filename* | committed) destination *destination-name*]

**Release Information** Statement introduced in Junos OS Release 8.4.
Statement introduced in Junos OS Release 9.0 for EX Series switches.
Support at the **[edit event-options policy *policy-name* then change-configuration]** hierarchy level introduced in Junos OS Release 12.1.

**Description** Associate a user with an action in an event policy. The event policy action is executed under the privileges of the associated user.

**Default** If you do not associate a user with an action, the action is executed as user **root**.

**Options** ***username***—Username that is configured at the **[edit system login]** hierarchy level.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Changing the User Privilege Level for an Event Policy Action on page 583

## within

**Syntax**

```
within seconds {
  events [ events ];
  not events [ events ];
  trigger (after | on | until) event-count;
}
```

**Hierarchy Level** [edit event-options policy *policy-name*]

**Release Information** Statement introduced in Junos OS Release 7.5.

**Description** Create a list of events that must (or must not) occur within a specified time interval for the policy to be triggered.

The statements are explained separately.

**Options** ***seconds***—Interval between events.

**Range:** 60 through 604,800 seconds

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using Correlated Events to Trigger an Event Policy on page 555

PART 6

# Event Automation

CHAPTER 32

# Event Scripts Overview

This chapter discusses the following topics:



## Event Scripts Overview



### Event Script Programming Overview

Junos OS event scripts are triggered automatically by defined event policies in response to a system event and can instruct Junos OS to take immediate action. Event scripts automate network and device management and troubleshooting. Event scripts can perform functions available through the remote procedure calls (RPCs) supported by either Junos XML management protocol or the Junos Extensible Markup Language (XML) API. Event scripts are executed by the event process (eventd).

Event scripts allow you to do the following:

- Automatically diagnose and fix problems in the network
- Monitor the overall status of a device.
- Run automatically as part of an event policy that detects periodic error conditions
- Change the configuration in response to a problem

Event scripts are based on the Junos XML management protocol and the Junos XML API, which are discussed in "Junos XML API and Junos XML Management Protocol Overview" on page 15. Event scripts can be written in either the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) scripting language. Event scripts use XPath to locate the operational objects to be inspected and XSLT constructs to specify the actions to perform on the located operational objects. The actions can change the output or execute additional commands based on the output. For more information about XPath and XSLT, see "XPath Overview" on page 22.

## How Event Scripts Work

Event scripts initiate operational commands when triggered by an event policy. When an event policy is triggered, this policy forwards event details to the event script. You enable event scripts by listing the names of one or more event script files within the **[edit event-options event-script]** hierarchy level. These scripts contain instructions that execute operational mode commands and inspect the output automatically. Event scripts are invoked within an event policy. For information about event policies, see "Event Notifications and Policies Overview" on page 549 and "Executing Event Scripts in an Event Policy" on page 578.

You can use event scripts to generate changes to the device configuration by including the **<load-configuration>** tag element. 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.

**Related Documentation**

- XSLT Overview on page 19
- SLAX Overview on page 35

CHAPTER 33

# Writing Event Scripts

This chapter explains how to write event scripts and includes the following topics:



## Required Boilerplate for Event Scripts

When you write event 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 event scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make 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: Understanding Extension Functions in the jcs and slax Namespaces" on page 95 and "Junos Script Automation: Named Templates in the jcs Namespace Overview" on page 124.

Event 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 event scripts that you create. The XSLT boilerplate follows:

**XSLT Boilerplate for Event 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">
 9      <event-script-results>
          <!-- ... Insert your code here ... -->
10      </event-script-results>
11    </xsl:template>
      <!-- ... insert additional template definitions here ... -->
12  </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"?>
```

Line 2 opens the style sheet and specifies the XSLT version as 1.0.

```
2  <xsl:stylesheet version="1.0"
```

Lines 3 through 6 list all the namespace mappings commonly used in event 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.

```
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 (in the file **/usr/libdata/cscript/import/junos.xsl**). 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" on page 124 and "Junos Named Templates in the jcs Namespace Summary" on page 125.

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

Line 8 defines a template that matches the **</>** element. The **<xsl:template match="/">** element is the root element and represents the top level of the XML hierarchy. All XML Path Language (XPath) expressions in the script must start at the top level. This allows the script to access all possible Junos XML and Junos XML protocol remote procedure calls (RPCs). For more information, see "XPath Overview" on page 22 and **xsl:template match="/" Template**.

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

After the **<xsl:template match="/">** tag element, the **<event-script-results>** and **</event-script-results>** container tags must be the top-level child tags, as shown in Lines 9 and 10.

```
9      <event-script-results>
       <!-- ... insert your code here ... -->
10     </event-script-results>
```

Line 11 closes the template.

```
11   </xsl:template>
```

Between Line 11 and Line 12, you can define additional XSLT templates that are called from within the **<xsl:template match="/">** template.

Line 12 closes the style sheet and the event script.

```
12  </xsl:stylesheet>
```

**SLAX Boilerplate for Event Scripts**

```
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 / {
  <event-script-results> {
    /*
    * Insert your code here
    */
  }
}
```

## Mapping Operational Mode Commands and Output Fields to Junos XML Notation

In event scripts, you use tag elements from the Junos XML API to represent operational mode commands and output fields. For the Junos XML equivalent of commands and output fields, consult the *Junos XML API Operational Reference*.

You can also display Junos XML by directing the output from the **show** command to the **| display xml** command:

```
user@host> operational-mode-command | display xml
```

For example:

```
user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
    <interface-information
xmlns="http://xml.juniper.net/junos/10.0RI0/junos-interface" junos:style="terse">
        <physical-interface>
            <name>dsc</name>
            <admin-status>up</admin-status>
            <oper-status>up</oper-status>
        </physical-interface>
        <physical-interface>
            <name>fxp0</name>
            <admin-status>up</admin-status>
            <oper-status>up</oper-status>
            <logical-interface>
                <name>fxp0.0</name>
                <admin-status>up</admin-status>
                <oper-status>up</oper-status>
                ...
```

## Using RPCs and Operational Mode Commands in Event Scripts

Most Junos operational mode commands have XML equivalents. These XML commands can be executed remotely using the *remote procedure call* (RPC) protocol. All operational

mode commands that have XML equivalents are listed in the *Junos XML API Operational Reference*.

RPC and operational mode command use in event scripts is discussed in more detail in the following sections:



## Using RPCs in Event Scripts

You can invoke remote procedure calls (RPCs) in event scripts. For each event script that invokes RPCs, you must include the **remote-execution** statement at the **[edit event-options event-script file *filename*]** hierarchy level. For each remote device where an RPC is executed, you must configure the SSH host key information for the that device on the local device where the event script is executed.

For each remote device where an RPC is executed, specify the device hostname and the corresponding username and passphrase at the **remote-execution** level of the configuration hierarchy.

```
[edit event-options event-script file filename]
remote-execution {
   remote-hostname {
      username username;
      passphrase passphrase;
   }
}
```

The remote hostnames and their corresponding username and passphrase, in addition to the event details, are passed as input to the event script when it is triggered by an event policy. For more information about the details that are forwarded to the event script, see "Capturing and Using Event Details and Remote Execution Details in Event Scripts" on page 671. A connection handle to the remote host is generated with the **jcs:open()** function using ***remote-hostname***, ***username***, and ***passphrase*** as arguments; for more information about this function, see **open()**. The following code obtains a connection handle for each remote host included in the configuration:

XSLT Syntax

```
<xsl:for-each select="event-script-input/remote-execution-details">
   <xsl:variable name="d" select="remote-execution-detail"/>
   <xsl:variable name="connection"
        select="jcs:open($d/remote-hostname,$d/username,$d/passphrase)"/>
   ...
</xsl:for-each>
```

SLAX Syntax

```
for-each (event-script-input/remote-execution-details) {
   var $d = remote-execution-detail;
   var $connection = jcs:open($d/remote-hostname,$d/username,$d/passphrase);
   ...
}
```

To execute an RPC on a remote device, an SSH session must be established. In order for the script to establish the connection, you must either configure the SSH host key

information for the remote device on the local device where the script will be executed, or the SSH host key information for the remote device must exist in the known hosts file of the user executing the script. For each remote device where the RPC is executed, configure the SSH host key information with one of the following methods:

- To configure SSH known hosts on the local device, include the **host** statement, and specify hostname and host key options for the remote device at the **[edit security ssh-known-hosts]** hierarchy level of the configuration.
- To manually retrieve SSH host key information, issue the **set security ssh-known-hosts fetch-from-server *hostname*** configuration mode command to instruct Junos OS to connect to the remote device and add the key.

  ```
  user@host# set security ssh-known-hosts fetch-from-server router2
  The authenticity of host 'router2 (10.10.10.1)' can't be established.
  RSA key fingerprint is 30:18:99:7a:3c:ed:40:04:0f:fd:c1:57:7e:6b:f3:90.
  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added 'router2,10.10.10.1' (RSA) to the list of known
  hosts.
  ```

- To manually import SSH host key information from a file, use the **set security ssh-known-hosts load-key-file *filename*** configuration mode command and specify the known-hosts file.

  ```
  user@host# set security ssh-known-hosts load-key-file /var/tmp/known_hosts
  Import SSH host keys from trusted source /var/tmp/known_hosts ? [yes,no] (no)
  yes
  ```

- Alternatively, the user executing the script can log in to the local device, SSH to the remote device, and then manually accept the host key, which is added to that user's known hosts file. In the following example, root is logged in to **router1**. In order to execute a remote RPC on **router2**, root adds the host key of **router2** by issuing the **ssh router2** operational mode command and manually accepting the key.

  ```
  root@router1> ssh router2
  The authenticity of host 'router2 (10.10.10.1)' can't be established.
  RSA key fingerprint is 30:18:99:7a:3c:ed:40:04:0f:fd:c1:57:7e:6b:f3:90.
  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added 'router2,10.10.10.1' (RSA) to the list of known
  hosts.
  ```

After configuring the required SSH host key and obtaining a connection handle to the remote device, the event script can execute RPCs with the **jcs:execute()** extension function on that remote device. For more information about this function, see execute(). To use an RPC in the event script, include the RPC in a variable declaration and execute it with the **jcs:execute()** function; the connection handle and RPC variable declaration are provided as arguments to the **jcs:execute()** function.

**XSLT Syntax**

```
<xsl:variable name="rpc">
  <get-interface-information/> # Junos RPC for the show interfaces command
</xsl:variable>
<xsl:variable name="out" select="jcs:execute($connection, $rpc)"/>
```

**SLAX Syntax**

```
var $rpc = <get-interface-information>;
var $out = jcs:execute($connection, $rpc);
```

where **connection** is the connection handle to the remote host. Any number of RPCs can be executed within the context of this connection handle until it is closed with the **jcs:close()** function.

## Displaying the RPC Tags for a Command

To display the remote procedure call (RPC) XML tags for an operational mode command, enter **display xml rpc** after the pipe symbol ( | ).

The following example displays the RPC tags for the **show route** command:

```
user@host> show route | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.1I0/junos">
    <rpc>
        <get-route-information>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

## Using Operational Mode Commands in Event Scripts

Some operational mode commands do not have XML equivalents. If a command is not listed in the *Junos XML API Operational Reference*, it does not have an XML equivalent.

Another way to determine whether a command has an XML equivalent is to issue the command followed by the **| display xml** command:

```
user@host> operational-mode-command | display xml
```

If the output includes only tag elements like **<output>**, **<cli>**, and **<banner>**, the command might not have an XML equivalent. In the following example, the output indicates that the **show host** command has no XML equivalent:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
   <output>
       ...
    </output>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

NOTE: For some commands that have an XML equivalent, the output of the piped **| display xml** command does not include tag elements other than **<output>**, **<cli>**, and **<banner>** only because the relevant feature is not configured. For example, the **show services cos statistics forwarding-class** command has an XML equivalent that returns output in the **<service-cos-forwarding-class-statistics>** response tag, but if the configuration does not include include any statements at the **[edit class-of-service]** hierarchy level then there is no actual data for the **show services cos statistics forwarding-class | display xml** command to display. The output is something like this:

```
user@host> show services cos statistics forwarding-class | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/8.3I0/junos">
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

For this reason, the information in the *Junos XML API Operational Reference* is normally more reliable.

An event script can include commands that have no XML equivalent. Use the **<command>**, **<xsl:value-of>**, and **<output>** elements in the script, as shown in the following code snippet. This snippet is expanded and fully described in "Example: Displaying DNS Hostname Information Using an Op Script" on page 507.

```
<xsl:variable name="query">
  <command>
    <xsl:value-of select="concat('show host ', $hostname)"/>
  </command>
</xsl:variable>
<xsl:variable name="result" select="jcs:invoke($query)"/>
<xsl:variable name="host" select="$result"/>
<output>
  <xsl:value-of select="concat('Name: ', $host)"/>
</output>
...
```

## Capturing and Using Event Details and Remote Execution Details in Event Scripts

When an event script is triggered by an event policy, the initiating event policy forwards a set of event details to the triggered event script. These event details can be captured, evaluated, and sent to log files as required. In addition, any configured remote execution details are also forwarded to the event script. The remote execution details allow the event script to invoke remote procedure calls as detailed in "Using RPCs and Operational Mode Commands in Event Scripts" on page 667.

Two types of event details are returned: triggered events and received events. *Triggered events* record the details of the event that triggered the policy. *Received events* record the details of events that happened before the triggering event. Event details and remote execution details are forwarded to the event script as XML in the following format:

```
<event-script-input>
  <junos-context>
    ...
  </junos-context>
  <trigger-event>
    <id>event-id</id>
    <type>event-type</type>
    <generation-time>timestamp</generation-time>
    <process>
      <name>process-name</name>
      <pid>pid</pid>
    </process>
    <hostname>hostname</hostname>
    <facility>facility-string</facility>
    <severity>severity-string</severity>
    <attribute-list>
      <attribute>
        <name>attribute-name</name>
        <value>attribute-value</value>
      </attribute>
    </attribute-list>
  </trigger-event>
  <received-events>
    <received-event>
      <id>event-id</id>
      <type>event-type</type>
      <generation-time>timestamp</generation-time>
      <process>
        <name>process-name</name>
        <pid>pid</pid>
      </process>
      <hostname>hostname</hostname>
      <facility>facility-string</facility>
      <severity>severity-string</severity>
      <attribute-list>
        <attribute>
          <name>attribute-name</name>
          <<value>attribute-value</value>
        </attribute>
      </attribute-list>
    </received-event>
  </received-events>
  <remote-execution-details>
    <remote-execution-detail>
      <remote-hostname>hostname</remote-hostname>
      <username>username</username>
      <passphrase>passphrase</passphrase>
    </remote-execution-detail>
  </remote-execution-details>
</event-script-input>
```

For information about the **<junos-context>** element, see "Junos Script Automation: Global Parameters and Variables in the junos.xsl File" on page 137.

For information about one method for using event details, see "Example: Limiting Event Script Output Based on a Specific Event Type" on page 683.

CHAPTER 34

# Configuring Event Scripts

Event scripts allow you to automate network troubleshooting and network management. This chapter discusses command-line interface (CLI) configuration statements and operational mode commands for enabling and executing scripts. Much of this discussion is applicable mainly to op scripts; however, most of the CLI statements discussed have a Junos XML protocol counterpart, and thus the concepts discussed in this section are helpful for event scripts.

To configure event scripts, include the following statements at the **[edit event-options]** hierarchy level:

```
[edit event-options]
event-script {
    file filename {
        checksum (md5 | sha-256 | sha1) hash;
        refresh;
        refresh-from url;
        remote-execution {
            remote-hostname {
                passphrase user-password;
                username user-login;
            }
        }
        source url;
    }
    max-datasize
    refresh;
    refresh-from url;
    traceoptions {
        file <filename> <files number> <size size> <world-readable | no-world-readable>;
        flag flag;
        no-remote-trace;
    }
}
```

This chapter discusses the following topics:

## Replacing an Event Script

You can update or replace an existing event script without changing the device's configuration or disrupting operations. Follow these steps:

1. Edit or write the new event script.
2. Copy the script to the **/var/db/scripts/event** directory on the hard drive or the **/config/scripts/event** directory on the flash drive; for information about setting the storage location for scripts, see "Storing Scripts in Flash Memory" on page 214. Only users who belong to the Junos **super-user** login class can alter files in these directories.

   NOTE: If the device has dual Routing Engines, remember to copy the script to the **/var/db/scripts/event** or **/config/scripts/event** directory on both Routing Engines. The **commit synchronize** command does not automatically copy scripts between Routing Engines.

3. Issue the **request system scripts event-scripts reload** operational mode command.

   ```
   user@host> request system scripts event-scripts reload
   ```

All event scripts are reloaded into the eventd process' memory.

## Enabling an Event Script

Event scripts are stored on a device's hard drive in the **/var/db/scripts/event** directory or on the flash drive in the **/config/scripts/event** directory. Only users in the Junos OS **super-user** login class can access and edit files in these directories. For information about setting the storage location for scripts, see "Storing Scripts in Flash Memory" on page 214.

NOTE: If the device has dual Routing Engines and you want to enable an event script to execute on both Routing Engines, you must copy the script to the **/var/db/scripts/event** or **/config/scripts/event** directory on both Routing Engines. The **commit synchronize** command does not automatically copy scripts between Routing Engines.

You must enable an event op script before it can be executed. Include the **file *filename*** statement at the **[edit event-options events-script]** hierarchy level, specifying the name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an event script. Only users who belong to the Junos **super-user** login class can enable event scripts.

```
[edit event-options event-script]
file filename;
```

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

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

## Configuring Checksum Hashes for an Event Script

You can configure one or more checksum hashes that can be used to verify the integrity of an event 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/even** 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/event/script1.slax) = 3af7884eb56e2d4489c2e49b26a39a97
   user@host> file checksum sha1 /var/db/scripts/commit/script1.slax
     SHA1 (/var/db/scripts/event/script1.slax) =
     00dc690fb08fb049577d012486c9a6dad34212c0
   user@host> file checksum sha-256 /var/db/scripts/commit/script1.slax
     SHA256 (/var/db/scripts/event/script1.slax) =
     150bf53383769f3bfedd41fe73320777f208d4fda81230cb27b8738
   ```

4. Configure the script.

   ```
   [edit event-options event-script]
   user@host# set file script1.slax checksum
     md5 3af7884eb56e2d4489c2e49b26a39a97
   [edit event-options event-script]
   user@host# set file script1.slax checksum
     sha-1 00dc690fb08fb049577d012486c9a6dad34212c0
   [edit event-options event-script]
   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 and the event policy fails.

**Related Documentation**

- Configuring Checksum Hashes for a Commit Script on page 282
- Configuring Checksum Hashes for an Op Script on page 475

- 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 an Event Script

When you issue the **commit** command, event scripts enabled at the **[edit event-options event-script]** hierarchy level are placed into system memory and enabled for execution. After the commit operation completes, an event script is executed in response to an event notification within an event policy. For more information, see "Executing Event Scripts in an Event Policy" on page 578.

## Tracing Event Script Processing

Event script tracing operations track all event 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 event script tracing is to log important events in a file called **escript.log** located in the **/var/log** directory. When the file **escript.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. The resulting files are **escript.log.0.gz**, then **escript.log.1.gz**, until there are 10 trace files. Then the oldest trace file (**escript.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 Enabling Traceoptions for Event Scripts

If no event script trace options are configured, the simplest way to view the trace output of an event script is to configure the **output** trace flag and issue the **show log escript.log | last** command. To do this, perform the following steps:

1. If you have not done so already, enable an event script by including the **file** statement at the **[edit event-options event-script]** hierarchy level:

   ```
   [edit event-options event-script]
   user@host# set file filename
   ```

2. Enable trace options by including the **traceoptions flag output** statement at the **[edit event-options event-script]** hierarchy level:

   ```
   [edit event-options event-script]
   user@host# set traceoptions flag output
   ```

3. Issue the **commit** command:

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

4. Display the resulting trace messages recorded in the **/var/log/escript.log** file. At the end of the log is the output generated by the event script you enabled in Step 1 after a configured event policy is triggered and invokes the script. To display the end of the log, issue the **show log escript.log | last** operational mode command:

```
[edit]
user@host# run show log escript.log | last
```

Table 43 on page 679 summarizes useful filtering commands that display selected portions of the **escript.log** file.

### Table 43: Event Script Tracing Operational Mode Commands

| Task | Command |
|---|---|
| Display logging data associated with all event script processing. | **show log escript.log** |
| Display processing for only the most recent operation. | **show log escript.log \| last** |
| Display processing for script errors. | **show log escript.log \| match error** |
| Display processing for a particular script. | **show log escript.log \| match *filename*** |

### Example: Minimum Configuration for Enabling Traceoptions for Event Scripts

Display the trace output of the event script file **source-route.xsl**:

```
[edit]
event-options {
   event-script {
      file source-route.xsl;
      traceoptions flag output;
   }
}
```

```
[edit]
user@host# commit
[edit]
user@host# run show log escript.log | last
```

## Configuring Tracing of Event 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 event-options event-script traceoptions]** hierarchy level:

```
[edit event-options event-script 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 Event Script Log Filename

By default, the name of the file that records trace output is **escript.log**. You can specify a different name by including the **file** statement at the **[edit event-options event-script traceoptions]** hierarchy level:

```
[edit event-options event-script traceoptions]
file filename;
```

## Configuring the Number and Size of Event 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 event-options event-script traceoptions file <*filename*>]** hierarchy level:

```
[edit event-options event-script 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 Event Script Log Files

By default, access to the event 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 event-options event-script traceoptions file <*filename*>]** hierarchy level.

```
[edit event-options event-script traceoptions file <filename>]
(world-readable | no-world-readable);
```

The **no-world-readable** statement restricts event script log access to the owner. The **world-readable** statement enables unrestricted access to the event script log file.

### Configuring the Event 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 event-options event-script traceoptions]** hierarchy level:

```
[edit event-options event-script traceoptions]
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
```

Table 44 on page 681 describes the meaning of the event script tracing flags.

**Table 44: Event Script Tracing Flags**

| Flag | Description | Default Setting |
|---|---|---|
| **all** | Trace all operations. | Off |
| **events** | Trace important events. | On |
| **input** | Trace event script input data. | Off |
| **offline** | Generate data for offline development. | Off |
| **output** | Trace event script output data. | Off |
| **rpc** | Trace event script RPCs. | Off |
| **xslt** | Trace the Extensible Stylesheet Language Transformations (XSLT) library. | Off |

CHAPTER 35

# Event Script Examples

This chapter provides the following sample event script:



## Example: Limiting Event Script Output Based on a Specific Event Type

In situations where an event policy is triggered by multiple event types, you can limit the number of events that trigger the event script. For example, the following event policy triggers the **event-details.slax** event script whenever a **ui_login_event** or **ui_logout_event** occurs.

```
event-options {
  policy event-detail {
    events [ ui_login_event ui_logout_event ];
    then {
      event-script event-details.slax {
        output-filename systemlog;
        destination /tmp;
      }
    }
  }
}
```

The **event-details.slax** event script writes a log file only when the **ui_login_event** event occurs.

```
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";
ns ext = "http://xmlsoft.org/XSLT/namespace";

var $event-definition = {
  <event-options> {
    <policy> {
      <namex> "event-detail";
      <eventsx> "ui_login_event";
      <thenx> {
        <event-scriptx> {
          <namex> "event_detail.slax";
          <output-filenamex> "foo";
```

```
            <destinationx> {
              <namex> "foo";
            }
          }
        }
      }
    }
  }

match / {
  <event-script-resultsx> {
    <event-triggered-this-policyx> {
      expr event-script-input/trigger-event/id;
    }
    <type-of-eventx> {
      expr event-script-input/trigger-event/type;
    }
    <process-namex> {
      expr event-script-input/trigger-event/attribute-list/attribute/name;
    }
  }
}
```

CHAPTER 36

# Summary of Event Script Configuration Statements

This chapter describes each configuration statement for event scripts. The statements are organized alphabetically.

## 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.
Statement introduced in Junos OS Release 11.1 for the QFX Series.

**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 282
- Configuring Checksum Hashes for an Event Script on page 677
- Configuring Checksum Hashes for an Op Script on page 475
- Executing an Op Script from a Remote Site on page 476
- 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*

## event-script (Event Options)

**Syntax**

```
event-script {
  file filename {
    checksum (md5 | sha-256 | sha1) hash;
    refresh;
    refresh-from url;
    remote-execution {
      remote-hostname {
        passphrase user-password;
        username user-login;
      }
    }
    source url;
  }
  max-datasize
  refresh;
  refresh-from url;
  traceoptions {
    file <filename> <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
  }
}
```

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** For Junos OS event scripts, configure scripting mechanisms.

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 on page 213

## file

**Syntax**

```
file filename {
  checksum (md5 | sha-256 | sha1) hash;
  refresh;
  refresh-from url;
  remote-execution {
    remote-hostname {
      passphrase user-password;
      username user-login;
    }
  }
  source url;
}
```

**Hierarchy Level** [edit event-options event-script]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** For Junos OS event scripts, enable an event script that is located in the **/var/db/scripts/event** directory.

**Options** *filename*—The name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an event script.

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**

- Enabling an Event Script on page 676

## max-datasize

**Syntax** max-datasize *size*;

**Hierarchy Level** [edit event-options event-script],
[edit system scripts commit],
[edit system scripts op]

**Release Information** Statement introduced in Junos OS Release 12.3.

**Description** Maximum amount of memory allocated for the data segment during execution of a script of the given type. Junos OS sets the maximum memory limit for the executing script to the configured value irrespective of the total memory available on the system at the time of execution. If the executing script exceeds the specified maximum memory limit for that script type, it exits gracefully.

**Default** If you do not include the **max-datasize** statement, the system allocates half of the total available memory of the system up to a maximum value of 128 MB for the data segment portion of the executed script.

**Options** ***size***—Maximum amount of memory allocated for the data segment during execution of a script of the given type. If you do not specify a unit of measure, the default is bytes.

**Syntax:** ***size*** to specify bytes, ***size*k** to specify KB, ***size*m** to specify MB, or ***size*g** to specify GB

**Range:** 2,3068,672 (22 MB) through 1,073,741,824 (1 GB)

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- max-policies on page 640
- Example: Configuring Limits on Executed Event Policies and Memory Allocation for Scripts on page 227

## refresh (Event Scripts)

**Syntax** refresh;

**Hierarchy Level** [edit event-options event-script],
[edit event-options event-script file *filename*]

**Release Information** Statement introduced in Junos OS Release 9.6.
Statement introduced in Junos OS Release 9.6 for EX Series switches.

**Description** For Junos OS event scripts, overwrite the local copy of all enabled event scripts or a single enabled script located in the **/var/db/scripts/event** directory with the copy located at the source URL, 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 on page 218
- refresh-from (Event Scripts) on page 690
- source (Event Policy) on page 692

## refresh-from (Event Scripts)

**Syntax** refresh-from *url*;

**Hierarchy Level** [edit event-options event-script],
[edit event-options event-script file *filename*]

**Release Information** Statement introduced in Junos OS Release 9.6.
Statement introduced in Junos OS Release 9.6 for EX Series switches.

**Description** For Junos OS event scripts, overwrite the local copy of all enabled event scripts or a single enabled script located in the **/var/db/scripts/event** directory with the copy located at a URL other than the URL specified in the **source** statement.

**Options** *url*—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 on page 222
- refresh (Event Scripts) on page 690
- source (Event Policy) on page 692

## remote-execution

**Syntax**

```
remote-execution {
  remote-hostname {
    passphrase user-password;
    username user-login;
  }
}
```

**Hierarchy Level** [edit event-options event-script file *filename*]

**Release Information** Statement introduced in Junos OS Release 9.6.
Statement introduced in Junos OS Release 9.6 for EX Series switches.

**Description** For Junos OS event scripts, enable event scripts to invoke RPCs on a local or remote host.

**Options**

**passphrase *user-password***—User's password for the remote host.

***remote-hostname***—Name of the remote host with which the event script will communicate.

**username *username***—User's login name for the remote host.

**Required Privilege Level** maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

**Related Documentation**

- Using RPCs and Operational Mode Commands in Event Scripts on page 667

## source (Event Policy)

**Syntax** source *url*;

**Hierarchy Level** [edit event-options event-script file *filename*]

**Release Information** Statement introduced in Junos OS Release 9.6.
Statement introduced in Junos OS Release 9.6 for EX Series switches.

**Description** For Junos OS event scripts, specify the location of the source file for an enabled script located in the **/var/db/scripts/event** directory. When you include the **refresh** statement at the same hierarchy level, the local copy is overwritten by the version stored at the specified URL.

**Options** *url*—Master source file for an event script 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**

- refresh (Event Scripts) on page 690
- refresh-from (Event Scripts) on page 690
- Using a Master Source Location for a Script on page 218

## traceoptions (Event Scripts)

**Syntax**

```
traceoptions {
  file <filename> <files number> <size size> <world-readable | no-world-readable>;
  flag flag;
  no-remote-trace;
}
```

**Hierarchy Level** [edit event-options event-script]

**Release Information** Statement introduced in Junos OS Release 7.6.
Statement introduced in Junos OS Release 9.0 for EX Series switches.

**Description** Define tracing operations for event scripts.

**Default** If you do not include this statement, no event script–specific tracing operations are performed.

**Options**

**file *filename***—Name of the file to receive the output of the tracing operation. All files are placed in the directory **/var/log**. By default, event script process tracing output is placed in the file **escript.log**. If you include the **file** statement, you must specify a filename. To retain the default, you can specify **escript.log** as the filename.

**Default:** /var/log/escript.log

**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 *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 event script input data
- **offline**—Generate data for offline development
- **output**—Log event script output data
- **rpc**—Log event 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:** *size* to specify bytes, *size*k to specify KB, *size*m to specify MB, or *size*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 Event Script Processing on page 678

PART 7

# Index

# Index

## Symbols



## A



## B

## C

## D

## E

## F

## G



## H



## I

## J



## K



## L

## R



## S

## T



## U

## V



## W



## X

# Index of Statements and Commands

## S



## T



## U



## W



## X