JUNOS® Internet Software

Configuration and Diagnostic Automation Guide

Release 9.5

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

Part Number: 530-029303-01, Revision 1

JUNOS® Internet Software Configuration and Diagnostic Automation Guide
Release 9.5

Writing: Tony Mauro, Michael Scruggs, Brenda Wilden
Editing: Sonia Saruba, Joanne McClintock, Nancy Kurahashi
Illustration: Faith Bradford
Cover Design: Edmonds Design

Revision History
13 April 2009—530-029303-01—Revision 1

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

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. The JUNOS software 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

READ THIS END USER LICENSE AGREEMENT ("AGREEMENT") BEFORE DOWNLOADING, INSTALLING, OR USING THE SOFTWARE. BY DOWNLOADING, INSTALLING, OR USING THE SOFTWARE OR OTHERWISE EXPRESSING YOUR AGREEMENT TO THE TERMS CONTAINED HEREIN, YOU (AS CUSTOMER OR IF YOU ARE NOT THE CUSTOMER, AS A REPRESENTATIVE/AGENT AUTHORIZED TO BIND THE CUSTOMER) CONSENT TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT OR CANNOT AGREE TO THE TERMS CONTAINED HEREIN, THEN (A) DO NOT DOWNLOAD, INSTALL, OR USE THE SOFTWARE, AND (B) YOU MAY CONTACT JUNIPER NETWORKS REGARDING LICENSE TERMS.

1. **The Parties.** The parties to this Agreement are (i) Juniper Networks, Inc. (if the Customer's principal office is located in the Americas) or Juniper Networks (Cayman) Limited (if the Customer's principal office is located outside the Americas) (such applicable entity being referred to herein as "Juniper"), and (ii) the person or organization that originally purchased from Juniper or an authorized Juniper reseller the applicable license(s) for use of the Software ("Customer") (collectively, the "Parties").

2. **The Software.** In this Agreement, "Software" means the program modules and features of the Juniper or Juniper-supplied software, for which Customer has paid the applicable license or support fees to Juniper or an authorized Juniper reseller, or which was embedded by Juniper in equipment which Customer purchased from Juniper or an authorized Juniper reseller. "Software" also includes updates, upgrades and new releases of such software. "Embedded Software" means Software which Juniper has embedded in or loaded onto the Juniper equipment and any updates, upgrades, additions or replacements which are subsequently embedded in or loaded onto the equipment.

3. **License Grant.** Subject to payment of the applicable fees and the limitations and restrictions set forth herein, Juniper grants to Customer a non-exclusive and non-transferable license, without right to sublicense, to use the Software, in executable form only, subject to the following use restrictions:

a. Customer shall use Embedded Software solely as embedded in, and for execution on, Juniper equipment originally purchased by Customer from Juniper or an authorized Juniper reseller.

b. Customer shall use the Software on a single hardware chassis having a single processing unit, or as many chassis or processing units for which Customer has paid the applicable license fees; provided, however, with respect to the Steel-Belted Radius or Odyssey Access Client software only, Customer shall use such Software on a single computer containing a single physical random access memory space and containing any number of processors. Use of the Steel-Belted Radius or IMS AAA software on multiple computers or virtual machines (e.g., Solaris zones) requires multiple licenses, regardless of whether such computers or virtualizations are physically contained on a single chassis.

c. Product purchase documents, paper or electronic user documentation, and/or the particular licenses purchased by Customer may specify limits to Customer's use of the Software. Such limits may restrict use to a maximum number of seats, registered endpoints, concurrent users, sessions, calls, connections, subscribers, clusters, nodes, realms, devices, links, ports or transactions, or require the purchase of separate licenses to use particular features, functionalities, services, applications, operations, or capabilities, or provide throughput, performance, configuration, bandwidth, interface, processing, temporal, or geographical limits. In addition, such limits may restrict the use of the Software to managing certain kinds of networks or require the Software to be used only in conjunction with other specific Software. Customer's use of the Software shall be subject to all such limitations and purchase of all applicable licenses.

d. For any trial copy of the Software, Customer's right to use the Software expires 30 days after download, installation or use of the Software. Customer may operate the Software after the 30-day trial period only if Customer pays for a license to do so. Customer may not extend or create an additional trial period by re-installing the Software after the 30-day trial period.

e. The Global Enterprise Edition of the Steel-Belted Radius software may be used by Customer only to manage access to Customer's enterprise network. Specifically, service provider customers are expressly prohibited from using the Global Enterprise Edition of the Steel-Belted Radius software to support any commercial network access services.

The foregoing license is not transferable or assignable by Customer. No license is granted herein to any user who did not originally purchase the applicable license(s) for the Software from Juniper or an authorized Juniper reseller.

4. **Use Prohibitions.** Notwithstanding the foregoing, the license provided herein does not permit the Customer to, and Customer agrees not to and shall not: (a) modify, unbundle, reverse engineer, or create derivative works based on the Software; (b) make unauthorized copies of the Software (except as necessary for backup purposes); (c) rent, sell, transfer, or grant any rights in and to any copy of the Software, in any form, to any third party; (d) remove any proprietary notices, labels, or marks on or in any copy of the Software or any product in which the Software is embedded; (e) distribute any copy of the Software to any third party, including as may be embedded in Juniper equipment sold in the secondhand market; (f) use any 'locked' or key-restricted feature, function, service, application, operation, or capability without first purchasing the applicable license(s) and obtaining a valid key from Juniper, even if such feature, function, service, application, operation, or capability is enabled without a key; (g) distribute any key for the Software provided by Juniper to any third party; (h) use the Software in any manner that extends or is broader than the uses purchased by Customer from Juniper or an authorized Juniper reseller; (i) use Embedded Software on non-Juniper equipment; (j) use Embedded Software (or make it available for use) on Juniper equipment that the Customer did not originally purchase from Juniper or an authorized Juniper reseller; (k) disclose the results of testing or benchmarking of the Software to any third party without the prior written consent of Juniper; or (l) use the Software in any manner other than as expressly provided herein.

5. **Audit.** Customer shall maintain accurate records as necessary to verify compliance with this Agreement. Upon request by Juniper, Customer shall furnish such records to Juniper and certify its compliance with this Agreement.

6. **Confidentiality.** The Parties agree that aspects of the Software and associated documentation are the confidential property of Juniper. As such, Customer shall exercise all reasonable commercial efforts to maintain the Software and associated documentation in confidence, which at a minimum includes restricting access to the Software to Customer employees and contractors having a need to use the Software for Customer's internal business purposes.

7. **Ownership.** Juniper and Juniper's licensors, respectively, retain ownership of all right, title, and interest (including copyright) in and to the Software, associated documentation, and all copies of the Software. Nothing in this Agreement constitutes a transfer or conveyance of any right, title, or interest in the Software or associated documentation, or a sale of the Software, associated documentation, or copies of the Software.

8. **Warranty, Limitation of Liability, Disclaimer of Warranty.** The warranty applicable to the Software shall be as set forth in the warranty statement that accompanies the Software (the "Warranty Statement"). Nothing in this Agreement shall give rise to any obligation to support the Software. Support services may be purchased separately. Any such support shall be governed by a separate, written support services agreement. TO THE MAXIMUM EXTENT PERMITTED BY LAW, JUNIPER SHALL NOT BE LIABLE FOR ANY LOST PROFITS, LOSS OF DATA, OR COSTS OR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THIS AGREEMENT, THE SOFTWARE, OR ANY JUNIPER OR JUNIPER-SUPPLIED SOFTWARE. IN NO EVENT SHALL JUNIPER BE LIABLE FOR DAMAGES ARISING FROM UNAUTHORIZED OR IMPROPER USE OF ANY JUNIPER OR JUNIPER-SUPPLIED SOFTWARE. EXCEPT AS EXPRESSLY PROVIDED IN THE WARRANTY STATEMENT TO THE EXTENT PERMITTED BY LAW, JUNIPER DISCLAIMS ANY AND ALL WARRANTIES IN AND TO THE SOFTWARE (WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE), INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT DOES JUNIPER WARRANT THAT THE SOFTWARE, OR ANY EQUIPMENT OR NETWORK RUNNING THE SOFTWARE, WILL OPERATE WITHOUT ERROR OR INTERRUPTION, OR WILL BE FREE OF VULNERABILITY TO INTRUSION OR ATTACK. In no event shall Juniper's or its suppliers' or licensors' liability to Customer, whether in contract, tort (including negligence), breach of warranty, or otherwise, exceed the price paid by Customer for the Software that gave rise to the claim, or if the Software is embedded in another Juniper product, the price paid by Customer for such other product. Customer acknowledges and agrees that Juniper has set its prices and entered into this Agreement in reliance upon the disclaimers of warranty and the limitations of liability set forth herein, that the same reflect an allocation of risk between the Parties (including the risk that a contract remedy may fail of its essential purpose and cause consequential loss), and that the same form an essential basis of the bargain between the Parties.

9. **Termination.** Any breach of this Agreement or failure by Customer to pay any applicable fees due shall result in automatic termination of the license granted herein. Upon such termination, Customer shall destroy or return to Juniper all copies of the Software and related documentation in Customer's possession or control.

10. **Taxes.** All license fees payable under this agreement are exclusive of tax. Customer shall be responsible for paying Taxes arising from the purchase of the license, or importation or use of the Software. If applicable, valid exemption documentation for each taxing jurisdiction shall be provided to Juniper prior to invoicing, and Customer shall promptly notify Juniper if their exemption is revoked or modified. All payments made by Customer shall be net of any applicable withholding tax. Customer will provide reasonable assistance to Juniper in connection with such withholding taxes by promptly: providing Juniper with valid tax receipts and other required documentation showing Customer's payment of any withholding taxes; completing appropriate applications that would reduce the amount of withholding tax to be paid; and notifying and assisting Juniper in any audit or tax proceeding related to transactions hereunder. Customer shall comply with all applicable tax laws and regulations, and Customer will promptly pay or reimburse Juniper for all costs and damages related to any liability incurred by Juniper as a result of Customer's non-compliance or delay with its responsibilities herein. Customer's obligations under this Section shall survive termination or expiration of this Agreement.

11. **Export.** Customer agrees to comply with all applicable export laws and restrictions and regulations of any United States and any applicable foreign agency or authority, and not to export or re-export the Software or any direct product thereof in violation of any such restrictions, laws or regulations, or without all necessary approvals. Customer shall be liable for any such violations. The version of the Software supplied to Customer may contain encryption or other capabilities restricting Customer's ability to export the Software without an export license.

12. **Commercial Computer Software.** The Software is "commercial computer software" and is provided with restricted rights. Use, duplication, or disclosure by the United States government is subject to restrictions set forth in this Agreement and as provided in DFARS 227.7201 through 227.7202-4, FAR 12.212, FAR 27.405(b)(2), FAR 52.227-19, or FAR 52.227-14(ALT III) as applicable.

13. **Interface Information.** To the extent required by applicable law, and at Customer's written request, Juniper shall provide Customer with the interface information needed to achieve interoperability between the Software and another independently created program, on payment of applicable fee, if any. Customer shall observe strict obligations of confidentiality with respect to such information and shall use such information in compliance with any applicable terms and conditions upon which Juniper makes such information available.

14. **Third Party Software.** Any licensor of Juniper whose software is embedded in the Software and any supplier of Juniper whose products or technology are embedded in (or services are accessed by) the Software shall be a third party beneficiary with respect to this Agreement, and such licensor or vendor shall have the right to enforce this Agreement in its own name as if it were Juniper. In addition, certain third party software may be provided with the Software and is subject to the accompanying license(s), if any, of its respective owner(s). To the extent portions of the Software are distributed under and subject to open source licenses obligating Juniper to make the source code for such portions publicly available (such as the GNU General Public License ("GPL") or the GNU Library General Public License ("LGPL")), Juniper will make such source code portions (including Juniper modifications, as appropriate) available upon request for a period of up to three years from the date of distribution. Such request can be made in writing to Juniper Networks, Inc., 1194 N. Mathilda Ave., Sunnyvale, CA 94089, ATTN: General Counsel. You may obtain a copy of the GPL at http://www.gnu.org/licenses/gpl.html, and a copy of the LGPL at http://www.gnu.org/licenses/lgpl.html.

15. **Miscellaneous.** This Agreement shall be governed by the laws of the State of California without reference to its conflicts of laws principles. The provisions of the U.N. Convention for the International Sale of Goods shall not apply to this Agreement. For any disputes arising under this Agreement, the Parties hereby consent to the personal and exclusive jurisdiction of, and venue in, the state and federal courts within Santa Clara County, California. This Agreement constitutes the entire and sole agreement between Juniper and the Customer with respect to the Software, and supersedes all prior and contemporaneous

agreements relating to the Software, whether oral or written (including any inconsistent terms contained in a purchase order), except that the terms of a separate written agreement executed by an authorized Juniper representative and Customer shall govern to the extent such terms are inconsistent or conflict with terms contained herein. No modification to this Agreement nor any waiver of any rights hereunder shall be effective unless expressly assented to in writing by the party to be charged. If any portion of this Agreement is held invalid, the Parties agree that such invalidity shall not affect the validity of the remainder of this Agreement. This Agreement and associated documentation has been written in the English language, and the Parties agree that the English version will govern. (For Canada: Les parties aux présentés confirment leur volonté que cette convention de même que tous les documents y compris tout avis qui s'y rattaché, soient redigés en langue anglaise. (Translation: The parties confirm that this Agreement and all related documentation is and will be in the English language)).

Abbreviated Table of Contents

		About This Guide	xxvii
Part 1		**Overview**	
	Chapter 1	Overview of Configuration and Diagnostic Automation	3
	Chapter 2	Scripts and Event Policy Configuration Statements	5
	Chapter 3	Introduction to the JUNOS XML and JUNOScript APIs	9
	Chapter 4	Understanding XSLT	15
	Chapter 5	Summary of XPath and XSLT Functions, Elements, Attributes, and Templates	27
	Chapter 6	Understanding SLAX	51
	Chapter 7	Summary of SLAX Statements	63
Part 2		**Commit Scripts**	
	Chapter 8	Commit Scripts Overview	79
	Chapter 9	Introduction to Writing Commit Scripts	91
	Chapter 10	Generating a Custom Warning, Error, or System Log Message	109
	Chapter 11	Summary of Message Tag Elements	123
	Chapter 12	Generating a Persistent or Transient Configuration Change	129
	Chapter 13	Creating Custom Configuration Syntax with Macros	145
	Chapter 14	Summary of XSLT Change Tag Elements	159
	Chapter 15	Configuring and Troubleshooting Commit Scripts	163
	Chapter 16	Summary of Commit Script Configuration Statements	179
	Chapter 17	Commit Script Examples	189
Part 3		**Operation (Op) Scripts**	
	Chapter 18	Operation (op) Scripts Overview	281
	Chapter 19	Introduction to Writing Operation Scripts	285
	Chapter 20	Configuring Operation Scripts	303
	Chapter 21	Operation Script Examples	315
	Chapter 22	Summary of Op Script Configuration Statements	335
Part 4		**Event Policy**	
	Chapter 23	Event Policy Overview	347
	Chapter 24	Configuring Event Policy	351

Chapter 25 Event Policy Examples 377
Chapter 26 Summary of Event Policy Configuration Statements 389

Part 5 Event Scripts

Chapter 27 Event Scripts Overview 415
Chapter 28 Introduction to Writing Event Scripts 419
Chapter 29 Configuring Event Scripts 439
Chapter 30 Event Script Examples 447
Chapter 31 Summary of Event Script Configuration Statements 449

Part 6 Index

Index 455
Index of Statements and Commands 463

Table of Contents

About This Guide **xxvii**

JUNOS Documentation and Release Notes xxvii
Objectives xxviii
Audience xxviii
Supported Platforms xxix
Using the Indexes xxix
Using the Examples in This Manual xxix
Merging a Full Example xxx
Merging a Snippet xxx
Documentation Conventions xxxi
Documentation Feedback xxxiii
Requesting Technical Support xxxiii

Part 1 Overview

Chapter 1 Overview of Configuration and Diagnostic Automation 3

Commit Scripts 3
Op Scripts 4
Event Scripts 4
Event Policies 4

Chapter 2 Scripts and Event Policy Configuration Statements 5

Any Hierarchy Level 5
[edit event-options] Hierarchy Level 6
[edit system scripts] Hierarchy Level 7
[edit event-options event-script] Hierarchy Level 8

Chapter 3 Introduction to the JUNOS XML and JUNOScript APIs 9

About XML 10
XML and JUNOScript Tag Elements 10
Document Type Definition 11
Advantages of Using the JUNOScript and JUNOS XML APIs 11
Overview of a JUNOScript Session 12

Chapter 4 Understanding XSLT 15

XPath 17
Templates 18
Unnamed Templates 19
Named Templates 19
Parameters 20
Variables 21
Programming Instructions 22
< xsl:choose > 23
< xsl:for-each select = "xpath-expression" > 23
< xsl:if test = "xpath-expression" > 24
Recursion 25
Context (Dot) 25

Chapter 5 Summary of XPath and XSLT Functions, Elements, Attributes, and Templates 27

XPath and XSLT Functions Shown in This Manual 28
concat() 28
contains() 28
count() 28
last() 29
name() 29
not() 29
position() 29
starts-with() 30
string-length() 30
substring-after() 30
substring-before() 31
XSLT Elements and Attributes Shown in This Manual 32
xsl:apply-templates 32
xsl:call-template 32
xsl:choose 33
xsl:comment 33
xsl:copy-of 34
xsl:element 34
xsl:for-each 35
xsl:if 35
xsl:import 36
xsl:otherwise 36
xsl:param 37
xsl:stylesheet 37
xsl:template 38
xsl:text 38
xsl:value-of 39
xsl:variable 39
xsl:when 40
xsl:with-param 40

JUNOS Extension Functions 41
jcs:break-lines() 41
jcs:empty() 41
jcs:first-of() 42
jcs:hostname() 42
jcs:invoke() 43
jcs:output() 43
jcs:printf() 44
jcs:progress() 44
jcs:regex() 45
jcs:sleep() 45
jcs:sysctl() 46
jcs:trace() 46
JUNOS Named Templates 47
jcs:edit-path 47
jcs:emit-change 48
jcs:emit-comment 49
jcs:statement 49

Chapter 6 Understanding SLAX 51

Overview 51
How SLAX Works 52
Manually Converting SLAX to XSLT and XSLT to SLAX 53
Statements 53
for-each Statement 53
if, else if, and else Statements 53
match Statement 54
ns Statement 55
version Statement 55
Elements 56
Expressions 56
Variables and Parameters 57
Attributes 58
Applying Templates 58
Template Parameters 59
Named Templates 59
Comments 60
Other XSLT Elements 61

Chapter 7 Summary of SLAX Statements 63

apply-templates 63
call 64
else 65
for-each 66
if 67
match 68
mode 69
param 70

priority 71
template 72
var 73
version 74
with 75

Part 2 Commit Scripts

Chapter 8 Commit Scripts Overview 79

Advantages of Using Commit Scripts 80
Storing the Commit Scripts 82
How Commit Scripts Work 82
Commit Script Input 83
Commit Script Output 84
Commit Scripts and the JUNOS Software Commit Model 85
Standard Commit Model 85
Commit Model with Commit Scripts 86
Using Multiple Commit Scripts 87
Using Large Commit Scripts 88
Using Commit Scripts 89

Chapter 9 Introduction to Writing Commit Scripts 91

Boilerplate for Commit Scripts 92
Importing the junos.xsl File 94
Extension Functions in the junos.xsl File 94
jcs:invoke() Function 95
jcs:progress() Function 95
jcs:output() Function 95
jcs:trace() Function 95
jcs:first-of() Function 95
jcs:printf() Function 96
jcs:sleep() Function 96
Templates in the junos.xsl File 96
<jcs:edit-path> Template 97
<jcs:emit-change> Template 97
<jcs:emit-comment> Template 100
<jcs:statement> Template 100
<xsl:template match="/"> Template 101
Design Considerations 103
Examples: Commit Scripts 104

Chapter 10 Generating a Custom Warning, Error, or System Log Message 109

Generating a Custom Warning, Error, or System Log Message 110
Examples: Generating a Custom Warning, Error, or System Log Message 113
Example: Generating a Custom Warning Message 113
Verifying the Commit Script Warning Output 114
Example: Generating a Custom Error Message 115
Verifying the Commit Script Error Output 116
Example: Generating a Custom System Log Message 118
Verifying the Commit Script Syslog Output 119
Message Tags 120

Chapter 11 Summary of Message Tag Elements 123

<syslog> 123
<xnm:error> 124
<xnm:warning> 126

Chapter 12 Generating a Persistent or Transient Configuration Change 129

Persistent and Transient Changes 130
Generating a Persistent or Transient Change 133
Examples: Generating a Persistent or Transient Change 137
Example: Generating a Persistent Change 137
Verifying the Commit Script Output 139
Example: Generating a Transient Change 140
Verifying the Commit Script Output 141
Removing a Persistent or Transient Change 142
Persistent and Transient Change Tags 143

Chapter 13 Creating Custom Configuration Syntax with Macros 145

How Macros Work 145
Creating a Custom Syntax 146
<data> Element 147
Expanding the Custom Syntax 148
Other Ways to Use Macros 151
Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements 152
Example: Creating Custom Configuration Syntax with Macros 153
Verifying the Commit Script Output 157

Chapter 14 Summary of XSLT Change Tag Elements 159

<change> 160
<transient-change> 161

Chapter 15 Configuring and Troubleshooting Commit Scripts 163

Adding and Removing Commit Scripts from the Configuration 163
Adding Commit Scripts to the Configuration 164
Removing Commit Scripts from the Configuration 165
Deactivating a Commit Script 165
Activating a Commit Script 165
Using Remote Commit Scripts 166
Refreshing Commit Script Files 166
Using the refresh-from Statement on a Single Commit Script 167
Using the refresh-from Statement Globally 167
Using the refresh Statement on a Single Commit Script 168
Using the refresh Statement Globally 168
Refreshing the Local Commit Script 168
Specifying a Remote Commit Script Source URL 169
Manually Converting a Script from XSLT to SLAX 169
Manually Converting a Script from SLAX to XSLT 170
Displaying Commit Script Output 170
Tracing Commit Script Processing 172
Minimum Configuration for Enabling and Viewing Traceoptions Output 172
Example: Minimum Configuration for Enabling and Viewing Traceoptions Output 173
Configuring Traceoptions 174
Configuring the Commit Script Log Filename 174
Configuring the Number and Size of Commit Script Log Files 174
Configuring the Trace Operations 175
Troubleshooting Commit Scripts 176

Chapter 16 Summary of Commit Script Configuration Statements 179

allow-transients 179
apply-macro 180
commit 181
direct-access 181
file 182
optional 182
refresh 183
refresh-from 183
scripts 184
source 185
traceoptions 186

Chapter 17 Commit Script Examples 189

Requiring and Restricting Configuration Statements 189
Testing ex-no-nukes.xsl 191
Requiring Internal Clocking on T1 Interfaces 192
Testing ex-clocking-error.xsl 194
Imposing a Minimum MTU Setting 195
Testing ex-so-mtu.xsl 196
Warning About a Deprecated Value 197
Testing ex-deprecated.xsl 198
Limiting the Number of E1 Interfaces 199
Testing ex-16-e1-limit.xsl 200
Limiting the Number of ATM Virtual Circuits 208
Testing ex-atm-vc-limit.xsl 209
Controlling IS-IS and MPLS Interfaces 211
Testing ex-iso.xsl 213
Adding T1 Interfaces to a RIP Group 214
Testing ex-rip-t1.xsl 216
Adding a Default Encapsulation Type 217
Testing ex-so-encap.xsl 218
Controlling LDP Configuration 220
Testing ex-ldp.xsl 223
Adding a Final then accept Term to a Firewall 224
Testing ex-add-accept.xsl 226
Configuring an Interior Gateway Protocol on an Interface 228
Testing ex-if-class.xsl 230
Creating a Complex Configuration Based on a Simple Interface Configuration 232
Testing ex-if-params.xsl 236
Configuring Administrative Groups for LSPs 238
Testing ex-lsp-admin.xsl 240
Controlling a Dual Routing Engine Configuration 242
Testing ex-dual-re.xsl and ex-dual-re2.xsl 245
Preventing Import of the Full Routing Table 246
Testing ex-import.xsl 248
Automatically Configuring Logical Interfaces and IP Addresses 249
Testing ex-atm-logical.xsl 254
Prepending a Global Policy 255
Testing ex-bgp-global-import.xsl 257
Assigning a Classifier 260
Testing ex-classifier.xsl 261
Loading a Base Configuration 263
Testing config-system.xsl 276

Part 3 Operation (Op) Scripts

Chapter 18 Operation (op) Scripts Overview 281

Op Script Programming 281
Operation (op) Scripts 282
Storing Op Scripts 282
How Op Scripts Work 282
Using Operation Scripts 283
Installing Op Scripts on a Router 283
Executing Op Scripts 284

Chapter 19 Introduction to Writing Operation Scripts 285

Boilerplate for Op Scripts 285
Displaying Operational Mode Fields in XML 287
Using RPCs and Operational Mode Commands 287
Importing the junos.xsl File 289
Extension Functions in the junos.xsl File 289
jcs:break-lines() Function 290
jcs:close() Function 290
jcs:dampen() Function 290
jcs:empty() Function 291
jcs:execute() Function 291
jcs:first-of() Function 292
jcs:getsecret() Function 292
jcs:hostname() Function 293
jcs:input() Function 293
jcs:invoke() Function 293
jcs:open() Function 294
jcs:output() Function 294
jcs:parse-ip() Function 294
jcs:printf() Function 295
jcs:progress() Function 295
jcs:regex() Function 295
jcs:sleep() Function 296
jcs:split() Function 296
jcs:sysctl() Function 296
jcs:syslog() Function 297
jcs:trace() Function 298
Templates in the junos.xsl File 298
<jcs:edit-path> Template 298
<jcs:emit-change> Template 299
<jcs:emit-comment> Template 301
<jcs:statement> Template 302

Chapter 20 Configuring Operation Scripts 303

Enabling an Operation Script and Defining a Script Alias 304
Executing an Operation Script 304
    Executing an Op Script 304
Declaring Arguments in Op Scripts 305
    Example: Declaring Arguments 306
Configuring Command-Line Help Text 307
    Examples: Configuring Command-Line Help Text 307
Specifying a Master Source for an Op Script 308
    Specifying a Master Source for an Op Script 308
Refreshing an Op Script from the Master Source 308
Refreshing an Op Script from a Different Location 309
Manually Converting a Script from XSLT to SLAX 310
Manually Converting a Script from SLAX to XSLT 310
Tracing Op Script Processing 311
    Minimum Configuration for Enabling and Viewing Traceoptions Output 311
        Example: Minimum Configuration for Enabling and Viewing Traceoptions Output 312
    Configuring Traceoptions 312
        Configuring the Op Script Log Filename 313
        Configuring the Number and Size of Op Script Log Files 313
        Configuring the Op Script Trace Operations 313

Chapter 21 Operation Script Examples 315

Restarting an FPC 315
    Testing ex-fpc.xsl 316
Displaying DNS Hostname Information 317
    Testing ex-hostname.xsl 319
Customizing Output of the show interfaces terse Command 320
    Line-by-Line Explanation of the Script 323
    Testing ex-interface.xsl 329
Finding LSPs to Multiple Destinations 330
    Testing ex-lsp.xsl 333
Importing and Exporting Files 333
    Exporting Files to a Remote Server 333
    Importing Files from a Remote Server 334

Chapter 22 Summary of Op Script Configuration Statements 335

arguments 335
command 335
description 336
file 337
op 338
refresh 339

refresh-from 339
scripts 340
source 341
traceoptions 342

Part 4 Event Policy

Chapter 23 Event Policy Overview 347

Introduction 347
How Event Policies Work 348

Chapter 24 Configuring Event Policy 351

Defining Destinations for File Archiving 353
Uploading Files 353
Executing Operational Mode Commands 355
Executing Event Scripts in an Event Policy 358
Correlating Events 363
Representing the Correlating Event in an Event Policy 365
Triggering a Policy Based on Event Count 366
Retrying the File Upload Action 366
Configuring an Event to Be Ignored 367
Using Regular Expressions to Refine the Set of Events That Cause a Policy to Be Executed 368
Associating an Optional User with an Event Policy Action 369
Assigning a Transfer Delay to an Event Policy Action 370
Generating Internal Events 371
Raising SNMP Traps 372
Referencing Nonstandard Events 373
Tracing Event Policy Processing 373
Configuring the Event Policy Log Filename 374
Configuring the Number and Size of Event Policy Log Files 374
Configuring Access to the Log File 375
Configuring a Regular Expression for Lines to Be Logged 375
Configuring the Trace Operations 375

Chapter 25 Event Policy Examples 377

Correlating Events Based on Receipt of Other Events Within a Specified Time Interval 377
Assigning a Transfer Delay to an Event Policy Action 378
Representing the Correlating Event in an Event Policy 380
Associating an Optional User with an Event Policy Action 381
Retrying the File Upload Action 382
Triggering a Policy Based on Event Count 383
Ignoring Events Based on Receipt of Other Events 385

Correlating Events Based on Event Attributes 385
Controlling Event Policy Using a Regular Expression 385
Generating an Internal Event Every Hour 386
Generating an Internal Event at Midnight 387
Dampening an Event 387
Raising an SNMP Trap in Response to an Event 388
Referencing Nonstandard Events 388

Chapter 26 Summary of Event Policy Configuration Statements 389

archive-sites 389
arguments 390
attributes-match 390
commands 391
destination 392
    destination (Command or Script Output) 392
    destination (Routing Platform Files) 392
destinations 393
equals 393
event-options 394
event-script 395
events 396
    events (Associating Events with a Policy) 396
    events (Correlating Events with Each Other) 396
execute-commands 397
filename 397
generate-event 398
ignore 398
matches 399
not 399
output-filename 400
output-format 400
policy 401
raise-trap 402
retry-count 402
starts-with 403
then 404
time-interval 405
time-of-day 405
traceoptions 406
transfer-delay 408
trigger 409
upload 410
    upload (Committed Configuration File) 410
    upload (Specified File) 410
user-name 411
within 411

Part 5 Event Scripts

Chapter 27 Event Scripts Overview 415

Event Script Programming 415
Event Scripts 416
Storing Event Scripts 416
How Event Scripts Work 416
Using Event Scripts 417
Installing Event Scripts on a Router 417
Replacing Event Scripts 417

Chapter 28 Introduction to Writing Event Scripts 419

Boilerplate for Event Scripts 419
Displaying Operational Mode Fields in XML 421
Using RPCs and Operational Mode Commands 422
Capturing and Using Event Details 423
Importing the junos.xsl File 425
Extension Functions in the junos.xsl File 425
jcs:break-lines() Function 426
jcs:close() Function 426
jcs:dampen() Function 426
jcs:empty() Function 427
jcs:execute() Function 427
jcs:first-of() Function 427
jcs:getsecret() Function 428
jcs:hostname() Function 428
jcs:input() Function 429
jcs:invoke() Function 429
jcs:open() Function 429
jcs:output() Function 430
jcs:parse-ip() Function 430
jcs:printf() Function 431
jcs:progress() Function 431
jcs:regex() Function 431
jcs:sleep() Function 432
jcs:split() Function 432
jcs:sysctl() Function 432
jcs:syslog() Function 432
jcs:trace() Function 434
Templates in the junos.xsl File 434
<jcs:edit-path> Template 434
<jcs:emit-change> Template 435
<jcs:emit-comment> Template 437
<jcs:statement> Template 438

Chapter 29 Configuring Event Scripts 439

Enabling an Event Script 439
Executing an Event Script 440
Declaring Arguments in Event Scripts 440
Example: Declaring Arguments 440
Manually Converting a Script from XSLT to SLAX 441
Manually Converting a Script from SLAX to XSLT 441
Tracing Event Script Processing 442
Minimum Configuration for Enabling and Viewing Traceoptions Output 442
Example: Minimum Configuration for Enabling and Viewing Traceoptions Output 443
Configuring Traceoptions 444
Configuring the Event Script Log Filename 444
Configuring the Number and Size of Event Script Log Files 444
Configuring the Event Script Trace Operations 445

Chapter 30 Event Script Examples 447

Limiting Event Script Output Based on a Specific Event Type 447

Chapter 31 Summary of Event Script Configuration Statements 449

event-script 449
file 450
traceoptions 451

Part 6 Index

Index 455
Index of Statements and Commands 463

List of Figures

Part 1 Overview

Chapter 4 Understanding XSLT 15

Figure 1: Flow of XSLT Script Through the XSLT Engine 16

Chapter 6 Understanding SLAX 51

Figure 2: SLAX Script Input and Output 52

Part 2 Commit Scripts

Chapter 8 Commit Scripts Overview 79

Figure 3: Commit Script Input and Output 83
Figure 4: Standard Commit Model 85
Figure 5: Commit Model with Commit Scripts Added 86
Figure 6: Configuration Evaluation by Multiple Commit Scripts 88

Chapter 9 Introduction to Writing Commit Scripts 91

Figure 7: Commit Script Input and Output 101

Chapter 13 Creating Custom Configuration Syntax with Macros 145

Figure 8: Macro Input and Output 145
Figure 9: Sample Macro and Corresponding JUNOS CLI Expansion 153

Part 3 Operation (Op) Scripts

Chapter 18 Operation (op) Scripts Overview 281

Figure 10: Op Script Input and Output 283

Part 4 Event Policy

Chapter 23 Event Policy Overview 347

Figure 11: Eventd Process Interaction with Other JUNOS Software Processes 348

List of Tables

About This Guide xxvii

Table 1: Additional Books Available Through http://www.juniper.net/books xxvii
Table 2: Notice Icons xxxi
Table 3: Text and Syntax Conventions xxxi

Part 1 Overview

Chapter 4 Understanding XSLT 15

Table 4: XSLT Concepts 16
Table 5: Variable Declarations: Examples and Pseudocode 22
Table 6: Programming Instructions: Examples and Pseudocode 24

Part 2 Commit Scripts

Chapter 10 Generating a Custom Warning, Error, or System Log Message 109

Table 7: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages 121

Chapter 12 Generating a Persistent or Transient Configuration Change 129

Table 8: Differences Between Persistent and Transient Changes 131
Table 9: Tags and Attributes for Creating Configuration Changes 143

Chapter 15 Configuring and Troubleshooting Commit Scripts 163

Table 10: Commit Script Configuration and Operational Mode Commands 171
Table 11: Commit Script Tracing Operational Mode Commands 173
Table 12: Commit Script Tracing Flags 175
Table 13: Troubleshooting Commit Scripts 176

Part 3 Operation (Op) Scripts

Chapter 19 Introduction to Writing Operation Scripts 285

Table 14: Facility Strings 297
Table 15: Severity String 298

Chapter 20 Configuring Operation Scripts 303

Table 16: Op Script Tracing Operational Mode Commands 312
Table 17: Op Script Tracing Flags 314

Part 4 Event Policy

Chapter 24 Configuring Event Policy 351

Table 18: Regular Expression Operators for the matches Statement 368
Table 19: Event ID by System Log Message Origin 373
Table 20: Event Policy Tracing Flags 376
Chapter 25 Event Policy Examples 377
Table 21: Event Count Triggers Policy 384

Part 5 Event Scripts

Chapter 28 Introduction to Writing Event Scripts 419
Table 22: Facility Strings 433
Table 23: Severity String 433
Chapter 29 Configuring Event Scripts 439
Table 24: Event Script Tracing Operational Mode Commands 443
Table 25: Event Script Tracing Flags 445

About This Guide

This preface provides the following guidelines for using the *JUNOS® Internet Software Configuration and Diagnostic Automation Guide*:

- JUNOS Documentation and Release Notes on page xxvii
- Objectives on page xxviii
- Audience on page xxviii
- Supported Platforms on page xxix
- Using the Indexes on page xxix
- Using the Examples in This Manual on page xxix
- Documentation Conventions on page xxxi
- Documentation Feedback on page xxxiii
- Requesting Technical Support on page xxxiii

JUNOS Documentation and Release Notes

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

If the information in the latest *JUNOS Release Notes* differs from the information in the documentation, follow the *JUNOS Release Notes*.

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

Table 1 on page xxvii lists additional books on Juniper Networks solutions that you can order through your bookstore. A complete list of such books is available at http://www.juniper.net/books.

Table 1: Additional Books Available Through http://www.juniper.net/books

Book	Description
Interdomain Multicast Routing	Provides background and in-depth analysis of multicast routing using Protocol Independent Multicast sparse mode (PIM SM) and Multicast Source Discovery Protocol (MSDP); details any-source and source-specific multicast delivery models; explores multiprotocol BGP (MBGP) and multicast IS-IS; explains Internet Gateway Management Protocol (IGMP) versions 1, 2, and 3; lists packet formats for IGMP, PIM, and MSDP; and provides a complete glossary of multicast terms.

Table 1: Additional Books Available Through http://www.juniper.net/books *(continued)*

Book	Description
JUNOS Cookbook	Provides detailed examples of common JUNOS software configuration tasks, such as basic router configuration and file management, security and access control, logging, routing policy, firewalls, routing protocols, MPLS, and VPNs.
MPLS-Enabled Applications	Provides an overview of Multiprotocol Label Switching (MPLS) applications (such as Layer 3 virtual private networks [VPNs], Layer 2 VPNs, virtual private LAN service [VPLS], and pseudowires), explains how to apply MPLS, examines the scaling requirements of equipment at different points in the network, and covers the following topics: point-to-multipoint label switched paths (LSPs), DiffServ-aware traffic engineering, class of service, interdomain traffic engineering, path computation, route target filtering, multicast support for Layer 3 VPNs, and management and troubleshooting of MPLS networks.
OSPF and IS-IS: Choosing an IGP for Large-Scale Networks	Explores the full range of characteristics and capabilities for the two major link-state routing protocols: Open Shortest Path First (OSPF) and IS-IS. Explains architecture, packet types, and addressing; demonstrates how to improve scalability; shows how to design large-scale networks for maximum security and reliability; details protocol extensions for MPLS-based traffic engineering, IPv6, and multitopology routing; and covers troubleshooting for OSPF and IS-IS networks.
Routing Policy and Protocols for Multivendor IP Networks	Provides a brief history of the Internet, explains IP addressing and routing (Routing Information Protocol [RIP], OSPF, IS-IS, and Border Gateway Protocol [BGP]), explores ISP peering and routing policies, and displays configurations for both Juniper Networks and other vendors' routers.
The Complete IS-IS Protocol	Provides the insight and practical solutions necessary to understand the IS-IS protocol and how it works by using a multivendor, real-world approach.

Objectives

This guide provides an overview, instructions for using, and examples of the commit script and self-diagnosis features of the JUNOS software. Commit scripts enforce custom configuration rules. They run at commit time and are based on Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) and two application programming interfaces (APIs): the JUNOS Extensible Markup Language (XML) API and the JUNOScript API. 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: This guide documents Release 9.5 of the JUNOS Internet software. For additional information about the JUNOS software—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, the JUNOS software currently supports the following platforms:

- J-series
- M-series
- MX-series
- T-series
- EX-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. 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 *JUNOS CLI User Guide*.

Documentation Conventions

Table 2 on page xxxi defines notice icons used in this guide.

Table 2: 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 3 on page xxxi defines the text and syntax conventions used in this guide.

Table 3: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: user@host> **configure**

Table 3: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
`Fixed-width text like this`	Represents output that appears on the terminal screen.	`user@host> show chassis alarms` `No alarms currently active`
Italic text like this	■ Introduces important new terms. ■ Identifies book names. ■ Identifies RFC and Internet draft titles.	■ A policy *term* is a named structure that defines match conditions and actions. ■ *JUNOS System Basics Configuration Guide* ■ 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: [edit] root@# **set system domain-name** *domain-name*
Plain text like this	Represents names of configuration statements, commands, files, and directories; IP addresses; 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. ■ The console port is labeled CONSOLE.
< > (angle brackets)	Enclose optional keywords or variables.	stub <default-metric *metric*>;
| (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 (*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]
		routing-options {
		  static {
		  route default {
		  nexthop *address*;
		  retain;
		  }
		  }
		}

; (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**. ■ To cancel the configuration, click **Cancel**.

Table 3: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
> (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 name
- Document part number
- Page number
- Software release version (not required for *Network Operations Guides [NOGs])*

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 postsales 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/customers/support/downloads/710059.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 located at 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, visit us at http://www.juniper.net/support/requesting-support.html.

Part 1
Overview

- Overview of Configuration and Diagnostic Automation on page 3
- Scripts and Event Policy Configuration Statements on page 5
- Introduction to the JUNOS XML and JUNOScript APIs on page 9
- Understanding XSLT on page 15
- Summary of XPath and XSLT Functions, Elements, Attributes, and Templates on page 27
- Understanding SLAX on page 51
- Summary of SLAX Statements on page 63

Chapter 1

Overview of Configuration and Diagnostic Automation

This chapter contains a brief overview of the configuration and diagnostic automation tools provided by the JUNOS software. These tools include commit scripts, operation and event scripts, and event policies.

This chapter discusses the following topics:

- Commit Scripts on page 3
- Op Scripts on page 4
- Event Scripts on page 4
- Event Policies on page 4

Commit Scripts

A commit script enforces custom configuration rules. Each time a new candidate configuration is committed, the script inspects the configuration. If a configuration violates your custom rules, the script corrects the problem by doing the following things:

- Generating custom error messages
- Generating custom warning messages
- Generating custom system log (syslog) messages
- Making changes to the configuration

For a more thorough overview, see “Commit Scripts Overview” on page 79.

Op Scripts

An op script automates network troubleshooting and network management by doing the following things:

- Customizing the output of operational mode commands.
- Ensuring your routing platform is configured to avoid or work around known problems in the JUNOS software.

For a more thorough overview, see "Operation (op) Scripts Overview" on page 281.

Event Scripts

An event script automates network troubleshooting and network management by doing the following things:

- Automatically diagnosing and fixing problems in your network.
- Monitoring the overall status of a routing platform.
- Ensuring your routing platform is configured to avoid or work around known problems in the JUNOS software.
- Running automatically as part of an event policy that detects periodic error conditions.
- Changing your configuration in response to a problem.

For a more thorough overview, see "Operation (op) Scripts Overview" on page 281.

Event Policies

An event policy is an if-then-else construct that defines actions to be executed by the software on receipt of a system log message. For each policy, you can configure multiple actions, as follows:

- Ignore the event.
- Upload a file to a specified destination.
- Execute JUNOS software operational mode commands.
- Execute JUNOS event scripts.

For a more thorough overview, see "Event Policy Overview" on page 347.

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 the JUNOS software, 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 on page 5
- [edit event-options] Hierarchy Level on page 6
- [edit system scripts] Hierarchy Level on page 7
- [edit event-options event-script] Hierarchy Level on page 8

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 event policy statements can be configured at the [edit event-options] hierarchy level:

```
event-options {
    destinations {
        destination-name {
            transfer-delay seconds;
            archive-sites {
                url password password;
            }
        }
    }
    generate-event event-name {
        time-interval seconds;
        time-of-day hh:mm:ss;
    }
    policy policy-name {
        events [ events ];
        within seconds not events [ events ];
        attributes-match {
            event1.attribute-name equals event2.attribute-name;
            event.attribute-name matches regular-expression;
            event1.attribute-name starts-with event2.attribute-name;
        }
        then {
            event-script filename {
                arguments {
                    parameter-name parameter-value;
                }
                output-filename filename;
                destination destination-name;
            }
            execute-commands {
                commands {
                    "command";
                }
                output-filename filename;
                output-format (text | xml);
                destination destination-name;
            }
            ignore;
            raise-trap;
            upload filename committed destination destination-name;
            upload filename filename destination destination-name;
        }
    }
    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 System Basics Configuration Guide*.

```
scripts {
    commit {
        allow-transients;
        direct-access;
        file filename {
            optional;
            refresh;
            refresh-from url;
            source url;
        }
        refresh;
        refresh-from url;
        traceoptions {
            file filename <files number> <size size>;
            flag flag;
        }
    }
    op {
        file filename {
            arguments {
                name {
                    description descriptive-text;
                }
            }
            command filename-alias;
            description descriptive-text;
            refresh;
            refresh-from url;
            source url;
        }
        refresh;
        refresh-from url;
        traceoptions {
            file filename <files number> <size size> <world-readable | no-world-readable>;
            flag flag;
        }
    }
}
```

[edit event-options event-script] Hierarchy Level

The following statements can be configured at the [edit event-options event-script] hierarchy level. This is not a comprehensive list of statements available at the [edit event-options event-script] hierarchy level. For more information about system configuration, see the *JUNOS System Basics Configuration Guide*.

```
event-script {
    file filename
    traceoptions {
        file filename <files number> <size size> <world-readable | no-world-readable>;
        flag flag;
    }
}
```

Chapter 3

Introduction to the JUNOS XML and JUNOScript APIs

The JUNOScript API (application programming interface) is an Extensible Markup Language (XML) application that client applications use to request and change configuration information on routing platforms that run the JUNOS software. The operations defined in the API are equivalent to configuration mode commands in the JUNOS command-line interface (CLI). Applications use the API 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 operations in the JUNOScript API 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 routing platform. The JUNOS XML API also includes tag elements that are the counterpart to JUNOS configuration statements.

Client applications request or change information on a routing platform by encoding the request with tag elements from the JUNOScript and JUNOS XML APIs and sending it to the JUNOScript server on the routing platform. (The JUNOScript server is integrated into the JUNOS software and does not appear as a separate entry in process listings.) The JUNOScript server directs the request to the appropriate software modules within the routing platform, encodes the response in JUNOScript and JUNOS XML tag elements, and returns the result to the client application. For example, to request information about the status of a routing platform's interfaces, a client application sends the <get-interface-information> tag element from the JUNOS XML API. The JUNOScript server gathers the information from the interface process and returns it in the <interface-information> tag element.

This manual explains how to use the JUNOScript and JUNOS XML APIs to configure Juniper Networks routing platforms or request information about configuration or operation. The main focus is on writing client applications to interact with the JUNOScript server, but you can also use the JUNOScript API to build custom end-user interfaces for configuration and information retrieval and display, such as a Web browser-based interface.

This chapter discusses the following topics:

- About XML on page 10
- Advantages of Using the JUNOScript and JUNOS XML APIs on page 11
- Overview of a JUNOScript Session on page 12

About XML

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.

The following sections discuss XML and JUNOScript tag elements:

- XML and JUNOScript Tag Elements on page 10
- Document Type Definition on page 11

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 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.

XML and JUNOScript Tag Elements

Items in an XML-compliant document or data set are always enclosed in paired opening and closing tags. 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:

```
<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 a JUNOScript or JUNOS XML 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 Juniper Networks documentation to indicate optional parts of CLI command strings.

JUNOScript and JUNOS XML tag elements obey the XML convention that the tag element name indicates the kind of information enclosed by the tag element. 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 routing platform, 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 manual conventionally uses just the name of the opening tag to represent the complete tag element (opening tag, contents, and closing tag). For example, it usually refers to the <input-bytes> tag element instead of the <input-bytes>*number-of-bytes*</input-bytes> tag element.

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.

Advantages of Using the JUNOScript and JUNOS XML APIs

The JUNOScript and JUNOS XML APIs are programmatic interfaces. They fully document all options for every supported JUNOS operational request and all elements in every JUNOS configuration statement. The tag names clearly indicate the function of an element in an operational request or 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. JUNOScript and JUNOS XML tag elements make it straightforward for client applications that request information from a routing platform to parse the output and find specific information.

The following example illustrates how the APIs make it easier to parse routing platform output and extract the needed information. It compares formatted ASCII and XML-tagged versions of output from a routing platform. The formatted ASCII follows:

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

This is the XML-tagged version:

```
<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, which is

```
, SNMP ifIndex
```

A problem arises if the format or ordering of output changes in a later version of the JUNOS software, for example, if a Logical index field is added following the interface index number:

```
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 following label instead:

```
, Logical index
```

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 JUNOScript 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.

Tagged output is also easier to transform into different display formats. For instance, you might want to display different amounts of detail about a given routing platform component at different times. When a routing platform returns formatted ASCII output, you have to design and write special routines and data structures in your display program to extract and store the information needed 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.

Overview of a JUNOScript Session

Communication between the JUNOScript server and a client application is session based. The two parties explicitly establish a connection before exchanging data and close the connection when they are finished. Each request from the client application and each response from the JUNOScript server constitutes a *well-formed* XML document, because the tag streams obey the structural rules defined in the JUNOScript and JUNOS XML DTDs for the kind of information they encode. Client applications must produce a well-formed XML document for each request by emitting tag elements in the required order and only in the legal contexts.

The following list outlines the basic structure of a JUNOScript session. For more specific information, see the *JUNOScript API Guide*.

1. The client application establishes a connection to the JUNOScript server and opens the JUNOScript session.
2. The JUNOScript server and client application exchange initialization information, used to determine if they are using compatible versions of the JUNOS software and the JUNOScript API.
3. The client application sends one or more requests to the JUNOScript server and parses its responses.
4. The client application closes the JUNOScript session and the connection to the JUNOScript server.

Chapter 4

Understanding XSLT

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

XSLT is a natural match for the JUNOS software, with its native XML capabilities. XSLT performs XML-to-XML transformations, turning one XML hierarchy into another. XSLT 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 pruning or adding) 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 the JUNOS software can easily express in XML) and for generating errors and warnings (which the JUNOS software communicates internally as XML). XSLT includes powerful mechanisms for finding configuration statements that match specific criteria. XSLT can then generate appropriate XML from these configuration statements to instruct the JUNOS user-interface (UI) components to perform the desired behavior. The JUNOScript application programming interface (API) defines XML elements for error, warning, and system log (syslog) messages.

Although XSLT provides a powerful scripting ability, its focus is specific and limited. It does not make the JUNOS software 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 routing platform is well-defined and can be viewed inside the command-line interface (CLI), using commands added for that purpose.

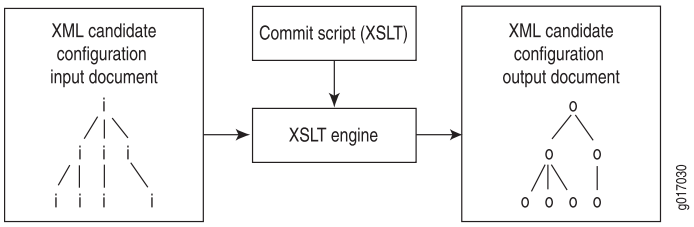
This chapter contains some overview material, intended as a brief introduction to XSLT. This chapter is not a comprehensive user guide for XSLT or XML Path Language (XPath). If you are already knowledgeable about XSLT, you can skip this chapter.

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 management process (mgd).

Figure 1 on page 16 shows the flow of an XSLT script through the XSLT engine.

Figure 1: Flow of XSLT Script Through the XSLT Engine



XSLT has seven basic concepts, summarized in Table 4 on page 16.

Table 4: XSLT Concepts

XSLT Concepts	Description
XPath	Expression syntax for specifying a node in the input document
Templates	Mechanisms for mapping input hierarchies to instructions that handle them
Parameters	A mechanism for passing arguments to templates
Variables	A mechanism for defining read-only references to nodes
Programming instructions	Mechanisms for defining logic in XSLT
Recursion	A mechanism by which templates call themselves to facilitate looping
Context (Dot)	The node currently being inspected in the input document

This chapter discusses the following topics:

- XPath on page 17
- Templates on page 18
- Parameters on page 20
- Variables on page 21
- Programming Instructions on page 22

- Recursion on page 25
- Context (Dot) on page 25

XPath

XSLT uses the 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.

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 commit scripts, the context node concept corresponds to JUNOS software 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 "Boilerplate for Commit Scripts" on page 92.

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. Following are several examples of path syntax:

- 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 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 must always be represented this way in XML.

For more information about XPath functions and operators, see "Summary of XPath and XSLT Functions, Elements, Attributes, and Templates" on page 27. We also recommend consulting a comprehensive XPath reference guide. XPath is fully described in the W3C specification at http://w3c.org/TR/xpath.

Templates

An XSLT script consists of one or more sets of rules called *templates*. Each template 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, and they are described in the following sections.

- Unnamed Templates on page 19
- Named Templates on page 19

Unnamed Templates

Unnamed 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 goes here ... - ->
</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. The select attribute can contain any XPath expression. If the <xsl:apply-templates> element has a select attribute, only nodes matching the XPath expression defined by the attribute are traversed. If the select attribute matches no nodes, nothing is traversed and nothing happens. Without a select attribute, all children of the context node are traversed.

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 where in the XML hierarchy a tag appears. 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 commit scripts, see "Importing the junos.xsl File" on page 94.

Named Templates

Named templates operate like functions in traditional programming languages, although with a verbose syntax. When markup grows complex, and when it appears in several different places in a style sheet, you can turn it into a named template.

Named templates resemble variables. However, they enable you to include data from the place where the template is applied, rather than merely inserting fixed text.

Parameters can be passed into named templates, and the parameters can be declared with default values. The following sample template named my-template 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 template is called without values for these parameters, the default values are used. As with unnamed templates, the select attribute can contain any XPath expression. If no select attribute is given for a parameter, it defaults to an empty value.

```
<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 goes here ... - ->
</xsl:template>
```

To invoke a named template, you must use the <xsl:call-template> element. It has a required name parameter that names the template it calls. When processed, the <xsl:call-template> element is replaced by the contents of the <xsl:template> element it names. 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 “Requiring and Restricting Configuration Statements” on page 189.

Parameters

Parameters can be passed to either named or unnamed templates using the <xsl:with-param> element. Inside the template, parameters must be declared and can then be referenced by prefixing their name with the dollar sign ($).

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 management process (mgd) to recursively apply templates to the configuration/system subtree. A parameter called host is passed to any templates that are processed.

```
<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.

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

To declare a default value for a parameter, include the select attribute and specify the desired default. If the template is invoked without the parameter, the XPath expression is evaluated and the results are assigned to the parameter.

The second template 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 other than the default source selected in the <report-changed> template.

```
<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>
```

Variables

You can define both local and global variables in XSLT. Variables are global if they are children of the <xsl:stylesheet> element. Otherwise, they are local. You can set the value of a variable only when you declare the variable by using the <xsl:variable> element. After that point, the value is fixed. The name attribute specifies the name of the variable. After declaring the variable, you can refer to it within an XPath expression using this name, prefixed with the $ character.

The following example declares the message variable. The message variable includes text and parameter values. The script 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 5 on page 22 provides examples of variable declarations and their pseudocode meanings.

Table 5: Variable Declarations: Examples and Pseudocode

Variable Declaration Examples	Pseudocode Meanings
<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 152.

Programming Instructions

XSLT has a number of traditional programming instructions. Their form tends to be verbose, because their syntax is built from XML elements. For summaries of all XSLT programming instructions used in this guide, see "Summary of XPath and XSLT Functions, Elements, Attributes, and Templates" on page 27.

The most important XSLT programming instructions for use with commit scripts, op scripts, and event scripts are as follows:

- <xsl:choose>***

- <xsl:for-each select="*xpath-expression*">
- <xsl:if test="*xpath-expression*">

The syntax and uses for these instructions are described in the following sections. Table 6 on page 24 provides examples of several instructions and their pseudocode meanings.

- < xsl:choose > on page 23
- < xsl:for-each select = "xpath-expression" > on page 23
- < xsl:if test = "xpath-expression" > on page 24

<xsl:choose>

The <xsl:choose> instruction is a 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. After the XSLT processor finds an XPath expression in an <xsl:when> element that evaluates as true, the XSLT processor ignores all subsequent <xsl:when> elements contained in the <xsl:choose> instruction, even if their XPath expressions evaluate as true. In other words, 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.

The <xsl:choose> instruction is similar to a switch statement in other programming languages, but the test expression can vary among <xsl:when> elements. The <xsl:when> element is the "case" of the switch statement. Any number of <xsl:when> elements can appear. 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 select="xpath-expression">

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

<xsl:if test="xpath-expression">

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">
</xsl:if>
```

Table 6: Programming Instructions: Examples and Pseudocode

Programming Instruction Examples	Pseudocode Meanings
<xsl:choose> <xsl:when test="system/host-name"> <change> <system> <host-name>M320</host-name> </system> </change> </xsl:when> <xsl:otherwise> <xnm:error> <message>Missing [edit system host-name] M320.</message> </xnm:error> </xsl:otherwise> </xsl:choose>	When the host-name statement is included at the [edit system] hierarchy level, change the hostname to M320. 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.
	
	In other words, match all apply-macro statements of the following form:
	
	apply-macro *apply-macro-name* {
	parameter-name;
	}
	
	And ignore all apply-macro statements of the form:
	
	apply-macro *apply-macro-name* {
	parameter-name parameter-value;
	}

<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.

Table 6: Programming Instructions: Examples and Pseudocode *(continued)*

Programming Instruction Examples	Pseudocode Meanings
<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.

Recursion

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 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 that 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). 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="count"/>
   </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; $max">
      <xsl:call-template name="count">
         <xsl:with-param name="cur" select="$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.

Context (Dot)

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

Summary of XPath and XSLT Functions, Elements, Attributes, and Templates

Extensible Markup Language Path Language (XPath) and Extensible Stylesheet Language Transformations (XSLT) together define multiple functions. In addition, XSLT defines multiple elements and attributes. All XPath and XSLT functions, elements, and attributes are supported by JUNOS commit scripts, op scripts, and event scripts. This chapter provides a partial reference guide, including only the XPath and XSLT functions, elements, and attributes shown in this manual.

This chapter also provides reference information for the extension functions and named templates available with JUNOS commit, op scripts, and event scripts. Extension functions allow you to perform operations that are difficult or impossible to perform in XPath. The named templates available with the JUNOS software also allow you to accomplish scripting tasks more easily. All extension functions and named templates available with JUNOS commit, op scripts, and event scripts are in the jcs: namespace, as indicated by the jcs: prefix.

This section is organized as follows:

- XPath and XSLT Functions Shown in This Manual on page 28
- XSLT Elements and Attributes Shown in This Manual on page 32
- JUNOS Extension Functions on page 41
- JUNOS Named Templates on page 47

XPath and XSLT Functions Shown in This Manual

The following sections explain each of the XPath and XSLT functions shown in this manual. The functions are organized alphabetically.

concat()

Syntax *string* concat(*string*, *string*+)

Description Return the concatenation of the arguments.

Usage Examples See "Limiting the Number of E1 Interfaces" on page 199, "Controlling IS-IS and MPLS Interfaces" on page 211, "Adding T1 Interfaces to a RIP Group" on page 214, "Configuring Administrative Groups for LSPs" on page 238, and "Controlling a Dual Routing Engine Configuration" on page 242.

contains()

Syntax *boolean* contains(*string*, *string*)

Description Return TRUE if the first argument string contains the second argument string, and otherwise returns FALSE.

Usage Examples See "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics starts-with() on page 30

count()

Syntax *number* count(*node-set*)

Description Return the number of nodes in the argument node-set.

Usage Examples See "Limiting the Number of E1 Interfaces" on page 199.

Related Topics last() on page 29, position() on page 29

last()

Syntax *number* last()

Description Return the index of the last node in the list that is currently being evaluated.

Usage Examples See "Limiting the Number of E1 Interfaces" on page 199.

Related Topics count() on page 28, position() on page 29

name()

Syntax *string* name(<*node-set*>)

Description Return the full name of the last node in the node set, including the prefix for its namespace declared in the source document. If no argument is passed, then returns the full name of the context node.

Usage Examples See "<jcs:emit-change> Template" on page 97.

not()

Syntax *boolean* not(*boolean*)

Description Return TRUE if its argument is FALSE, and FALSE if the argument is TRUE.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Controlling IS-IS and MPLS Interfaces" on page 211, "Adding a Default Encapsulation Type" on page 217, "Controlling LDP Configuration" on page 220, "Adding a Final then accept Term to a Firewall" on page 224, "Configuring Administrative Groups for LSPs" on page 238, "Controlling a Dual Routing Engine Configuration" on page 242, and "Preventing Import of the Full Routing Table" on page 246.

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 "Adding a Final then accept Term to a Firewall" on page 224 and "Prepending a Global Policy" on page 255.

Related Topics count() on page 28, last() on page 29

starts-with()

Syntax *boolean* starts-with(*string*, *string*)

Description Return TRUE if the first argument string starts with the second argument string, and returns FALSE otherwise.

Usage Examples See "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Limiting the Number of ATM Virtual Circuits" on page 208, "Adding T1 Interfaces to a RIP Group" on page 214, "Adding a Default Encapsulation Type" on page 217, and "Controlling a Dual Routing Engine Configuration" on page 242.

Related Topics contains() on page 28

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 "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

substring-after()

Syntax *string* substring-after(*string*, *string*)

Description Return the substring of the first argument string that occurs after the second argument string. If the second string is not contained in the first string, or if the second string is empty, then it returns an empty string.

Usage Examples See "Limiting the Number of E1 Interfaces" on page 199 and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics substring-before() on page 31

substring-before()

Syntax *string* substring-before(*string*, *string*)

Description Return the substring of the first argument string that occurs before the second argument string. If the second string is not contained in the first string, or if the second string is empty, then it returns an empty string.

Usage Examples See "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics substring-after() on page 30

XSLT Elements and Attributes Shown in This Manual

The following sections explain each of the XSLT elements shown in this manual. The elements are organized alphabetically.

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. The nodes to which the processor applies templates are selected by the path specified by the select attribute. 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—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 "Adding a Final then accept Term to a Firewall" on page 224 and "Preventing Import of the Full Routing Table" on page 246.

Related Topics xsl:call-template, xsl:for-each, xsl:template, xsl:with-param

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 are used to define parameters that are passed to the template.

Attributes name—Specifies the name of the called template.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Warning About a Deprecated Value" on page 197, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:apply-templates, xsl:template

xsl:choose

Syntax

```
<xsl:choose>
   <xsl:when test="boolean-expression">
   ...
   </xsl:when>
   <xsl:otherwise>
    ...
   </xsl:otherwise>
</xsl:call-template>
```

Description Express multiple conditional tests. 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.

Usage Examples See "Controlling a Dual Routing Engine Configuration" on page 242, "Preventing Import of the Full Routing Table" on page 246, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:if, xsl:otherwise, xsl:when

xsl:comment

Syntax

```
<xsl:comment>
   ...
</xsl:comment>
```

Description Generate a comment 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 < !- - ... Insert your comment here ... - - > comments, but these are ignored by the processor. To generate a comment within the final document, use an <xsl:comment> element.

Usage Examples See "Adding a Final then accept Term to a Firewall" on page 224.

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—Specifies an expression to select nodes to be copied.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189.

Related Topics xsl:value-of

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. To do this, enclose an XML element in curly brackets ({ }), as in <xsl:element name="{$isis-level-1}".

Usage Examples See "Creating a Complex Configuration Based on a Simple Interface Configuration" on page 232.

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 instance of identical XML elements. The element nodes are selected by the expression defined by the select attribute. Each of the nodes is then processed by the instructions contained in the <xsl:for-each> instruction.

Attributes select—Specifies an expression to select nodes to be processed.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Adding T1 Interfaces to a RIP Group" on page 214, "Configuring Administrative Groups for LSPs" on page 238, and "Controlling a Dual Routing Engine Configuration" on page 242.

Related Topics xsl:apply-templates

xsl:if

Syntax

```
<xsl:if test="boolean-expression">
    ...
</xsl:if>
```

Description Include a conditional construct that causes instructions to be processed if the Boolean expression held in the test attribute evaluates to TRUE.

Attributes test—Specifies a Boolean expression.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Limiting the Number of E1 Interfaces" on page 199, "Adding T1 Interfaces to a RIP Group" on page 214, and "Controlling a Dual Routing Engine Configuration" on page 242.

Related Topics xsl:choose, xsl:when

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 all examples listed in "Commit Script Examples" on page 189.

Related Topics xsl:stylesheet

xsl:otherwise

Syntax

```
<xsl:otherwise>
   ...
</xsl:otherwise>
```

Description Within an <xsl:choose> instruction, include the instructions that are processed if none of the expressions defined in the test attributes of the <xsl:when> elements evaluate as TRUE.

Usage Examples See "Controlling a Dual Routing Engine Configuration" on page 242 and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:choose, xsl:when

xsl:param

Syntax

```
<xsl:param name="qualified-name" select="expression">
   ...
</xsl:param>
```

Description Declare a parameter for a template (if it is within a template) or for the style sheet as a whole (if it is at the top level of the style sheet).

Attributes name—Defines the name of the parameter.

select—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. The select attribute or the content of the <xsl:param> element can define the default value. 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 "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Limiting the Number of ATM Virtual Circuits" on page 208, and "Preventing Import of the Full Routing Table" on page 246.

Related Topics xsl:template, xsl:variable, xsl:with-param

xsl:stylesheet

Syntax

```
<xsl:stylesheet version="1.0">
   <xsl:import href="../import/junos.xsl"/>
     ...
</xsl:stylesheet>
```

Description Include the document element for the style sheet. Contains all the top-level elements such as global variable and parameter declarations, import elements, and templates. 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.

Attributes version—Specifies the version of XSLT that is being used. The JUNOS software supports XSLT version 1.0.

Usage Examples See all examples listed in "Commit Script Examples" on page 189.

Related Topics xsl:import

xsl:template

Syntax

```
<xsl:template match="pattern" mode="qualified-name" name="qualified-name">
   <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 the XML document For example, match="/" matches the whole document.

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.

You can pass templates parameters by 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—Applies the template to nodes by specifying a pattern against which nodes are matched.

mode—Indicates the mode in which a template needs to be applied for the template to be used.

name—Calls the template by name.

Usage Examples See all examples listed in "Commit Script Examples" on page 189.

Related Topics xsl:apply-templates, xsl:call-template

xsl:text

Syntax

```
<xsl:text>
   ...
</xsl:text>
```

Description Insert literal text in the output.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Controlling IS-IS and MPLS Interfaces" on page 211, and "Adding a Final then accept Term to a Firewall" on page 224.

xsl:value-of

Syntax

```
<xsl:value-of select="string-expression"/>
```

Description Extract data from the XML structure. The select attribute specifies the expression that is evaluated. In the string 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—Specifies the expression that is evaluated.

Usage Examples See "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Controlling IS-IS and MPLS Interfaces" on page 211, "Configuring Administrative Groups for LSPs" on page 238, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:copy-of

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 covering 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—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 "Limiting the Number of E1 Interfaces" on page 199, "Limiting the Number of ATM Virtual Circuits" on page 208, "Configuring Administrative Groups for LSPs" on page 238, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:param

xsl:when

Syntax

```
<xsl:when test="boolean-expression">
    ...
</xsl:when>
```

Description Within an <xsl:choose> instruction, specify a set of processing that occurs when the expression specified in the test attribute evaluates as TRUE. 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.

Attributes test—Specifies a Boolean expression.

Usage Examples See "Controlling a Dual Routing Engine Configuration" on page 242, "Preventing Import of the Full Routing Table" on page 246, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:choose, xsl:if, xsl:otherwise

xsl:with-param

Syntax

```
<xsl:with-param name="qualified-name" select="expression">
    ...
</xsl:with-param>
```

Description Specify the value of a parameter to be passed into a template. It can be used when applying templates with the <xsl:apply-templates> instruction or calling templates with the <xsl:call-template> instruction.

Attributes name—Specifies the name of the parameter for which the value is being passed.

select—Determines 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 some 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 "Controlling a Dual Routing Engine Configuration" on page 242, "Preventing Import of the Full Routing Table" on page 246, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics xsl:apply-templates, xsl:call-template, xsl:param

JUNOS Extension Functions

The following sections explain each of the extension functions available when you write JUNOS commit scripts, op scripts, and event scripts. The functions are organized alphabetically.

jcs:break-lines()

Syntax jcs:break-lines(*expression*)

Description Break a simple element into multiple elements, delimited by new lines. This is especially useful for large <output> elements, such as those produced by the show pfe commands.

Usage Examples

```
var $lines = jcs:break-lines($output);
```

jcs:empty()

Syntax jcs:empty(*node-set*)
jcs:empty(*string*)

Description Return TRUE if the node set or string arguments are empty.

Usage Examples

```
if(jcs:empty($set)) { .... }
```

jcs:first-of()

Syntax jcs:first-of(*object*,+ "*expression*")

Description Return the first nonempty (non-null) item in a list.

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" )
```

The following example selects the description of a logical interface if there is a logical interface description. If not, it selects the description of the (parent) physical interface if there is a physical interface description. If not, it selects the concatenation of the physical interface name with a " ." and the logical unit number.

```
<xsl:variable name="description"
      select="jcs:first-of(description, ../description, concat(../name, '.', name))"/>
```

Related Topics Displaying DNS Hostname Information on page 317

jcs:hostname()

Syntax jcs:hostname(*expression*)

Description Return the fully qualified domain name of an address or hostname.

Usage Examples

```
<xsl:variable name="name" select="jcs:hostname($dest)"/>
<xsl:value-of select="concat($address, ' is ', jcs:hostname($address))"/>
```

Related Topics Finding LSPs to Multiple Destinations on page 330

jcs:invoke()

Syntax jcs:invoke(*rpc*)

Description Invoke a remote procedure call (RPC). It can be called with one argument, either a string containing a JUNOS XML or JUNOScript RPC method name or a tree containing an RPC. The result is the contents of the <rpc-reply> element, not the <rpc-reply> tag element itself.

Usage Examples In the following example, there is a test to see if the interface argument is included on the command line when the script is executed. If it is, the operational mode output of the show interfaces terse command is narrowed to include information about that interface only.

```
<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 an RPC without modifying the output:

```
<xsl:variable name="sw" select="jcs:invoke('get-software-information')"/>
```

Related Topics Customizing Output of the show interfaces terse Command on page 320

jcs:output()

Syntax jcs:output(*expression*)

Description Generate unformatted output text. The function emits an <output> element. The text appears in the CLI.

Usage Examples

```
<xsl:variable name="ignore" select="jcs:output('The VPN is up.')"/>
```

jcs:printf()

Syntax jcs:printf('*expression*')

Description Generate formatted output text. The text appears in the CLI. Most standard printf formats are supported, in addition to some JUNOS software-specific ones.

The %j1 operator emits the field only if the field was changed from the last time the function was run.

The %jc operator capitalizes the first letter of the format output.

The %jt{TAG} operator emits the tag if the field is not empty.

Usage Examples

```
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ –> }s\n',
                                 'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/>
```

jcs:progress()

Syntax jcs:progress('*expression*')

Description Issue a progress message back to the client (CLI) containing the single argument.

Usage Examples

```
<xsl:variable name="ignore" select="jcs:progress('Working...')"/>
```

jcs:regex()

Syntax jcs:regex(*expression*, *string*)

Description Return the set of strings matched by the given regular expression. This function requires two arguments, the regular expression and the string to match.

Usage Examples

```
var $pat = "([0-9]+)(:*)([a-z]*)";
var $a = jcs:regex($pat, "123:xyz");
var $b = jcs:regex($pat, "r2d2");
var $c = jcs:regex($pat, "test999!!!");
$a[1] == "123:xyz"    # full string that matches the regex
$a[2] == "123"        # ([0-9]+)
$a[3] == ":"          # (:*)
$a[4] == "xyz"        # ([a-z]*)
$b[1] == "2d"         # full string that matches the regex
$b[2] == "2"          # ([0-9]+)
$b[3] == ""           # (:*)   [empty match]
$b[4] == "d"          # ([a-z]*)
$c[1] == "999"        # full string that matches the regex
$c[2] == "999"        # ([0-9]+)
$c[3] == ""           # (:*)   [empty match]
$c[4] == ""           # ([a-z]*)   [empty match]
```

jcs:sleep()

Syntax jcs:sleep(*seconds* <, *milliseconds*>)

Description Cause the script to sleep for a specified number of seconds and optionally, milliseconds. You can use this function to help determine how a routing component is working over time. To do this, write a script that issues a command, calls the jcs:sleep() function, and reissues the same command.

Usage Examples In this example, jcs:sleep(1) means 1 second and jcs:sleep(0, 10) means 10 milliseconds.

```
<xsl:variable name="ignore" select="jcs:sleep(1)"/>
<xsl:variable name="ignore" select="jcs:sleep(0, 10)"/>
```

jcs:sysctl()

Syntax jcs:sysctl("*expression*", "*i*")
jcs:sysctl("*expression*", "s")

Description Return the value of the given expression or object as a string or an integer. Use the "i" argument to specify an integer. Use the "s" argument to specify a string.

Usage Examples

```
var $value = jcs:sysctl("kern.hostname", "s");
```

jcs:trace()

Syntax jcs:trace('*expression*')

Description Issue a trace message, which is sent to the trace file.

Usage Examples

```
<xsl:variable name="ignore" select="jcs:trace('test')"/>
```

JUNOS Named Templates

The following sections explain each of the named templates available when you write JUNOS commit scripts, op scripts, and event scripts. The templates are organized alphabetically.

To use these templates in your scripts, you simply include <xsl:call-template name=" *name*" > elements and pass in any required or optional parameters. The name attribute specifies the name of the called template.

jcs:edit-path

Syntax

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

Description Generate an <edit-path> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. By default, the location of the configuration error is passed as dot into the <jcs:edit-path> template. This location defaults to " ." , the current position in the XML hierarchy. You can alter the default by including the select attribute of the dot parameter.

Parameters <xsl:param name="dot" select="."><—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current context " ." as a default value. If you want to change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.

Usage Guidelines See " < jcs:edit-path > Template" on page 97.

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189.

Related Topics xsl:param, xsl:with-param

jcs:emit-change

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>...</xsl:text>
   </xsl:with-param>
   <xsl:with-param name="name" select="name($dot)"/>
   <xsl:with-param name="tag" select="'change'"/>
   <xsl:with-param name="tag" select="'transient-change'"/>
</xsl:call-template>
```

Description Generate a <change> or <transient-change> element, which results in a persistent or transient change to the configuration.

Parameters <xsl:param name="content">—Allows you to include the content of the change, relative to dot.

<xsl:param name="dot" select=".">—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current context " ." as a default value. If you want to change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.

<xsl:param name="message">—Allows you to include a warning message to be displayed by 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, include the dot parameter.

<xsl:param name="name" select="name($dot)"/>—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 name of the element in $dot (which in turn defaults to " ." , the current element).

<xsl:param name="tag" select="'change'"/>—Allows you to specify the type of change to be generated. By default, the <jcs:emit-change> template generates a permanent change, as designated by the 'change' expression. To specify a transient change, you must include the tag parameter and include the 'transient-change' expression, as shown here:

```
<xsl:with-param name="tag" select="'transient-change'"/>
```

Usage Guidelines See " <jcs:emit-change> Template" on page 97.

Usage Examples See "Imposing a Minimum MTU Setting" on page 195.

Related Topics xsl:param, xsl:with-param

jcs:emit-comment

Syntax

```
<junos:comment>
   <xsl:text>...</xsl:text>
</junos:comment>
```

Description Emit a simple comment. 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 Guidelines See "<jcs:emit-comment> Template" on page 100.

Usage Examples See "Adding a Final then accept Term to a Firewall" on page 224.

Related Topics jcs:emit-change on page 48

jcs:statement

Syntax

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

Description Generate a <statement> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The parameter dot can be passed into the <jcs:statement> template if the error is not at the current position in the XML hierarchy.

Parameters <xsl:param name="dot" select="."/>—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current context " ." as a default value. If you want to change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.

Usage Guidelines See "<jcs:statement> Template" on page 100.

Usage Examples See "Controlling a Dual Routing Engine Configuration" on page 242.

Related Topics xsl:param, xsl:with-param

Chapter 6

Understanding SLAX

This chapter discusses the following topics:

- Overview on page 51
- How SLAX Works on page 52
- Manually Converting SLAX to XSLT and XSLT to SLAX on page 53
- Statements on page 53
- Elements on page 56
- Expressions on page 56
- Variables and Parameters on page 57
- Attributes on page 58
- Applying Templates on page 58
- Template Parameters on page 59
- Named Templates on page 59
- Comments on page 60
- Other XSLT Elements on page 61

Overview

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

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 programming instructions and XPath expressions 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

For examples of commit and op scripts written in SLAX, see "Commit Script Examples" on page 189 and "Operation Script Examples" on page 315.

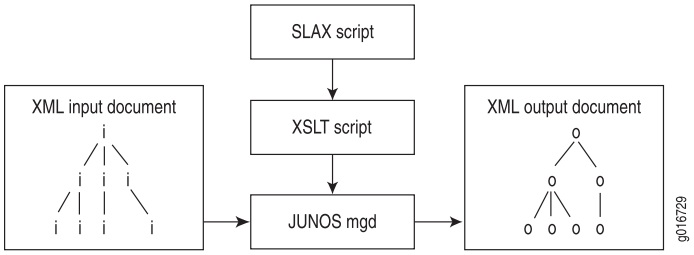
How SLAX Works

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.

SLAX functions as a preprocessor for XSLT. The JUNOS software automatically converts SLAX programming instructions (such as if, then, and else statements) into the equivalent XSLT instructions (such as <xsl:choose> and <xsl:if> elements). After this conversion, the XSLT transformation engine—which, for the JUNOS software, is the JUNOS management process (mgd)—is invoked.

Figure 2 on page 52 shows the flow of SLAX script input and output.

Figure 2: SLAX Script Input and Output



Manually Converting SLAX to XSLT and XSLT to SLAX

Converting scripts manually and studying the results facilitates learning of the differences between the two languages. If you have existing XSLT commit, op, and event scripts, conversion to SLAX allows C and PERL programmers to more easily read and maintain the scripts.

To convert XSLT scripts into SLAX, issue the request system scripts convert xslt-to-slax operational mode command, specifying a source file and a destination.

To convert SLAX scripts into XSLT, issue the request system scripts convert slax-to-xslt operational mode command, specifying a source file and a destination.

For more information, see "Manually Converting a Script from XSLT to SLAX" on page 169 or "Manually Converting a Script from XSLT to SLAX" on page 310 and "Manually Converting a Script from SLAX to XSLT" on page 170 or "Manually Converting a Script from SLAX to XSLT" on page 310.

Statements

This section lists some common SLAX statements, with brief examples and XSLT equivalents. For a complete list of SLAX statements, see "Summary of SLAX Statements" on page 63.

for-each Statement

The SLAX for-each statement functions like the <xsl:for-each> element. The statement consists of the for-each keyword, the parentheses-delimited expression, and a curly braces-delimited block.

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

The XSLT equivalent:

```
<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 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 */
}
```

Depending on the presence of the **else** clause, an **if** statement is transformed into either an <xsl:if> element or 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:

```
<xsl:choose>
   <xsl:when select="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:

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

```
        <message> ...</message>
    </xnm:error>
</xsl:template>
```

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 stylesheet 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 stylesheet, 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:

```
<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 stylesheets must begin with a version statement, which specifies the version number for the SLAX language. The current version is 1.0. SLAX version 1.0 uses XML version 1.0 and XSLT version 1.1.

```
version 1.0;
```

The XSLT equivalent:

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

Elements

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

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

The XSLT equivalent:

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

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

Expressions

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 an <xsl:text> or <xsl:value-of> element.

You encode strings using quotation marks (single or double). The concatenation operator is 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.

```
<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 equivalent XSLT:

```
<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>
```

Variables and Parameters

You use the var and param statements to declare variables and parameters. In SLAX, the variable name contains the dollar sign even in the declaration, unlike the name attribute of <xsl:variable> and <xsl:parameter>.

```
param $fido;
var $bone;
```

The XSLT equivalent:

```
<xsl:parameter name="fido"/>
<xsl:variable name="bone"/>
```

You can declare an initial value by following the variable name with an equal sign (=) and an expression.

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

The XSLT equivalent:

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

Attributes

Attributes of elements follow the style of XML. The attribute name is followed by an equal sign (=) and the value of the attribute.

```
<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 (_).

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

The XSLT equivalent:

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

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:

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

Applying Templates

You apply match templates using the apply-templates statement. This statement accepts an optional XPath expression, which is equivalent to the select attribute in an <xsl:apply-templates> element.

```
match configuration {
    apply-templates system/host-name;
}

match host-name {
    <hello> .;
}
```

The XSLT equivalent:

```
<xsl:template match="configuration">
   <xsl:apply-templates select="system/host-name"/>
</xsl:template>
<xsl:template match="host-name">
   <hello>
      <xsl:value-of select="."/>
   </hello>
</xsl:template>
```

Template Parameters

You can pass parameters to match templates using the with statement. The with statement consists of the keyword with and the name of the parameter, optionally followed by an equal sign (=) and a value expression. If you do not specify a value, the current value of the variable or parameter is passed.

```
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:

```
<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>
```

Named Templates

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 an optional equal sign (=) and value expression. You can declare additional parameters inside the block using the param statement.

You invoke named templates using the call statement, which consists of the call keyword followed by a set of parameter bindings. These bindings are a comma-separated list of parameter names, optionally followed by an equal sign (=) and a value expression. If you do not provide a value, the current value of the variable or parameter is passed. You can supply additional template parameters inside the block using the with statement.

```
match configuration {
   var $name-servers = name-servers/name;
   call temp:ting();
   call temp:ting($name-servers, $size = count($name-servers));
   call temp:ting() {
      with $name-servers;
      with $size = count($name-servers);
   }

    template temp:ting($name-servers, $size = 0) {
      <output> "template called with size " _ $size;
   }
}
```

The XSLT equivalent:

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

Comments

You enter comments in SLAX in the traditional C style, beginning with /* and ending with */.

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

The XSLT equivalent:

```
<!- -/*
* This is a comment
*/ - ->
```

Other XSLT Elements

Some XSLT elements are not directly translated into SLAX statements. Some examples of XSLT elements for which there are no SLAX equivalents 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 .;
    }
  }
}
```

Chapter 7

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.

apply-templates

Syntax apply-templates *expression*;

Description Apply one or more templates, according to the value of the node-set expression. The templates that are applied are passed the parameters specified by the with statement within the apply-templates statement.

Attributes *expression*—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 “Adding a Final then accept Term to a Firewall” on page 224 and “Preventing Import of the Full Routing Table” on page 246.

Related Topics match on page 68, mode on page 69, with on page 75

call

Syntax

```
call template-name (parameter-name = value) {
   /* code */
}
```

Description

Invoke a template. You can include a comma-separated list of parameters, with the parameter name and an optional equal sign (=) and value expression. If the value is not given, the current value of the parameter is passed.

You can declare additional parameters inside the code block using the with 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 “Requiring and Restricting Configuration Statements” on page 189, “Imposing a Minimum MTU Setting” on page 195, “Warning About a Deprecated Value” on page 197, and “Automatically Configuring Logical Interfaces and IP Addresses” on page 249.

Related Topics template on page 72, with on page 75

else

Syntax

```
else {
   /* code */
}
else {
   if (expression) {
      /* code */
   }
}
```

Description Include the instructions that are processed if none of the expressions defined in the test attributes of the if statement evaluate as TRUE.

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 “Controlling a Dual Routing Engine Configuration” on page 242 and “Automatically Configuring Logical Interfaces and IP Addresses” on page 249.

Related Topics if on page 67

for-each

Syntax

```
for-each (expression) {
   /* code */
}
```

Description Include a looping mechanism that repeats script processing for each instance of identical XML elements. The element nodes are selected by the *expression* attribute. Each of the nodes is then processed by the instructions contained in the for-each statement.

Attributes *expression*—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

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

XSLT Equivalent

```
<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>
```

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Adding T1 Interfaces to a RIP Group" on page 214, "Configuring Administrative Groups for LSPs" on page 238, and "Controlling a Dual Routing Engine Configuration" on page 242.

if

Syntax

```
if (expression) {
   /* code */
}
```

Description Include a conditional construct that causes instructions to be processed if the Boolean expression held in the test attribute evaluates to TRUE.

Attributes *expression*—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

```
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 "Controlling a Dual Routing Engine Configuration" on page 242, "Preventing Import of the Full Routing Table" on page 246, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics else on page 65

match

Syntax

```
match expression {
statements;
}
```

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 whole document.

Attributes *expression*—Specifies a pattern against which nodes are matched.

SLAX Example

```
match host-name {
   <hello> .;
}
```

XSLT Equivalent

```
<xsl:template match="host-name">
   <hello>
      <xsl:value-of select="."/>
   </hello>
</xsl:template>
```

Usage Examples See all examples listed in "Commit Script Examples" on page 189.

Related Topics apply-templates on page 63, mode on page 69

mode

Syntax mode *qualified-name*;

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.

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 “Adding a Final then accept Term to a Firewall” on page 224.

Related Topics apply-templates on page 63, match on page 68

param

Syntax param $*name*=*value*;

Description Declare a parameter for a template (within a template) or for the script as a whole (at the top level of the script). You can include an initial value by following the variable name with an equal sign (=) and a value expression.

In SLAX, parameter and variable names contain the dollar sign ($) even in the declaration. This is unlike the name attribute of <xsl:variable> and <xsl:parameter> elements.

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:parameter name="vrf"/>
<xsl:parameter name="dot" select="."/>
```

Usage Examples See "Requiring and Restricting Configuration Statements" on page 189, "Imposing a Minimum MTU Setting" on page 195, "Limiting the Number of E1 Interfaces" on page 199, "Limiting the Number of ATM Virtual Circuits" on page 208, and "Preventing Import of the Full Routing Table" on page 246.

Related Topics var on page 73

priority

Syntax priority *number*;

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 Topics apply-templates on page 63, match on page 68

template

Syntax

```
template qualified-name (parameter-name = value) {
    /* code */
}
```

Description Declare a 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 all examples listed in "Commit Script Examples" on page 189.

Related Topics call on page 64, with on page 75

var

Syntax var $*name*=*value*;

Description Declare a local or global variable. If the var statement appears at the top of the script, it is a global variable with a scope covering the entire script. Otherwise, it is a local variable. You can include an initial value by following the variable name with an equal sign (=) and a value expression.

In SLAX, parameter and variable names contain the dollar sign ($) even in the declaration. This is unlike the name attribute of <xsl:variable> and <xsl:parameter> elements.

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 "Limiting the Number of E1 Interfaces" on page 199, "Limiting the Number of ATM Virtual Circuits" on page 208, "Configuring Administrative Groups for LSPs" on page 238, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics param on page 70

version

Syntax version 1.0;

Description Specify the version of SLAX that is being used. All SLAX stylesheets 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. The JUNOS software supports SLAX version 1.0.

SLAX Example

```
version 1.0;
```

XSLT Equivalent

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

Usage Examples See all examples listed in "Commit Script Examples" on page 189.

with

Syntax with *name* = *value*;

Description Specify a variable or parameter to be passed into a template. You can use this statement when you apply templates with the apply-templates statement or call templates with the match statement.

Optionally, you can specify a value for the parameter by including an equal sign (=) and a value expression. If no value is given, the current value of the variable or parameter is passed.

Attributes *name*—Specifies the name of the variable or parameter for which the value is being passed.

value—Determines the value of the parameter.

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 "Controlling a Dual Routing Engine Configuration" on page 242, "Preventing Import of the Full Routing Table" on page 246, and "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

Related Topics apply-templates on page 63, match on page 68

Part 2
Commit Scripts

- Commit Scripts Overview on page 79
- Introduction to Writing Commit Scripts on page 91
- Generating a Custom Warning, Error, or System Log Message on page 109
- Summary of Message Tag Elements on page 123
- Generating a Persistent or Transient Configuration Change on page 129
- Creating Custom Configuration Syntax with Macros on page 145
- Summary of XSLT Change Tag Elements on page 159
- Configuring and Troubleshooting Commit Scripts on page 163
- Summary of Commit Script Configuration Statements on page 179
- Commit Script Examples on page 189

Chapter 8

Commit Scripts Overview

JUNOS commit scripts enforce custom configuration rules. Each time a new candidate configuration is committed, the active commit scripts are called and inspect the new candidate configuration. If a configuration violates your custom rules, the script can instruct the JUNOS software to perform various actions, including the following:

- Generate custom error messages.
- Generate custom warning messages.
- Generate custom system log (syslog) messages.
- Make changes to the configuration.

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

To view the router'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 scripts are based on the JUNOS XML application programming interface (API) and the JUNOScript API. For more information on the JUNOS XML API and the JUNOScript API, see “Introduction to the JUNOS XML and JUNOScript APIs” on page 9. 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 located configuration objects. The actions can change the configuration or generate messages about it. For more information on XSLT, see “Understanding XSLT” on page 15. For more information on SLAX, see “Understanding SLAX” on page 51.

This chapter discusses the following topics:

- Advantages of Using Commit Scripts on page 80
- Storing the Commit Scripts on page 82
- How Commit Scripts Work on page 82
- Using Commit Scripts on page 89

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 router 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 Label Distribution Protocol (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 Intermediate System-to-Intermediate System (IS-IS) routers are never enabled.

When a candidate configuration does not adhere to your design rules, a commit script can instruct the JUNOS software 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 Multiprotocol Label Switching (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.

All of these example scenarios are included in "Commit Script Examples" on page 189.

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 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 "Generating a Persistent or Transient Configuration Change" on page 129.

Storing the Commit Scripts

By default, commit, operation, and event scripts are stored on the router's hard drive. However, you can save these scripts to the flash memory by including the load-scripts-from-flash statement at the [edit system scripts] hierarchy level:

```
[edit system scripts]
load-scripts-from-flash;
```

The load-scripts-from-flash statement applies to all commit, operation, and event scripts, and changes the physical location of these scripts. This statement does not affect the script's operation. Commit scripts located on the router's hard drive are accessed from the /var/db/scripts/commit directory. Commit scripts located on the router's flash drive are accessed from the /config/scripts/commit directory.

You can view the currently active scripts on the router by viewing the scripts located in the /var/run/scripts/commit directory on the router. This directory allows you to view the active scripts without having to search through the configuration.

NOTE: When you switch from storing the files on the hard drive to storing files on the flash drive, or vice versa, you need to manually move any scripts residing in the former memory location to the new memory location. If you do not move these scripts, the scripts will not be available to the router.

How Commit Scripts Work

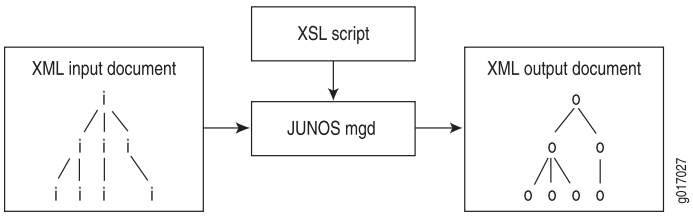
You enable commit scripts by listing the names of one or more commit script files at the [edit system scripts commit] hierarchy level. These scripts contain instructions that enforce custom configuration rules. Commit scripts are invoked during the commit process before the standard JUNOS validity checks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

When you perform a commit operation, the JUNOS software executes each script in turn, passing the configuration through the scripts. The script inspects the configuration, performs the necessary tests and validations, and generates a set of instructions, requesting the software to perform 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 routing platform configuration.

Figure 3 on page 83 shows the flow of commit script input and output.

Figure 3: Commit Script Input and Output



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

- Commit Script Input on page 83
- Commit Script Output on page 84
- Commit Scripts and the JUNOS Software Commit Model on page 85
- Using Multiple Commit Scripts on page 87
- Using Large Commit Scripts on page 88

Commit Script Input

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

After you issue a commit command, the JUNOS software automatically generates the candidate configuration in XML format and reads it into the management process (mgd), 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 routing platform.

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.

  For more information about the <jcs:emit-change> template, see "<jcs:emit-change> Template" on page 97.

The JUNOS software processes this output and performs the appropriate actions. Errors and warnings are passed back to the JUNOS CLI or to a JUNOScript 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.

Commit Scripts and the JUNOS Software Commit Model

The JUNOS software uses a commit model to update the router's configuration. This model allows you to make a series of changes to a candidate configuration without affecting the operation of the routing platform. When the changes are complete, you can commit the configuration. The commit operation saves the candidate configuration changes into the current configuration.

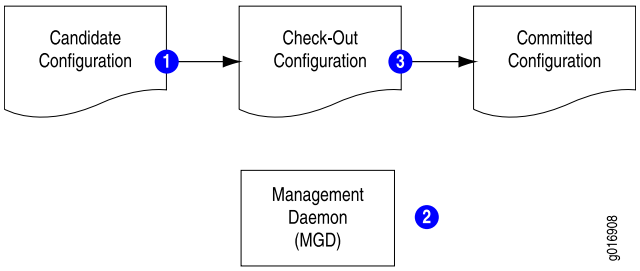
When you commit a set of changes in the candidate configuration, two methods are used to forward these changes to the current configuration:

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

Standard Commit Model

In the standard commit model, the mgd validates the candidate configuration based on the JUNOS validation rule. If this configuration file passes validation, then the file becomes the current configuration. Figure 4 on page 85 and the accompanying flow discussion show how the standard commit model works:

Figure 4: Standard Commit Model



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

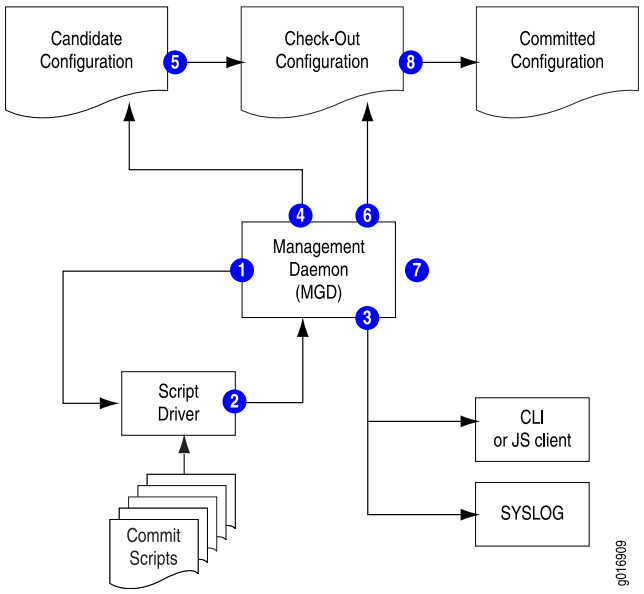
1. When the candidate configuration is committed, it is copied to become the checkout configuration.
2. The mgd validates the checkout configuration.
3. If no error occurs, the checkout configuration is copied as the current configuration.

Commit Model with Commit Scripts

By adding commit scripts to the commit model, the mgd validates the candidate configuration based on the JUNOS validation rule. If this configuration file passes validation, then the file becomes the current configuration.

When commit scripts are added to the standard commit model, the process becomes more complex. The mgd first passes an XML-formatted checkout configuration to a script driver. This driver handles the verification of the checkout configuration by the commit scripts. When verification is complete, the script driver returns an action XML file to the mgd for processing. The mgd processes this action XML file, updating the candidate and checkout configurations, issuing messages to the CLI, and writing information to the Syslog as required. Once the action XML file is processed, the standard JUNOS validation is handled by the mgd. Figure 5 on page 86 and the accompanying flow discussion show this process.

Figure 5: Commit Model with Commit Scripts Added



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

1. When the candidate configuration is committed, the mgd sends the XML-formatted candidate configuration to the script driver.
2. Each configured commit script is invoked against the candidate configuration, and each script can issue a set of actions for the mgd to perform.
3. The mgd processes the <error>, <warning>, and <syslog> actions within the action XML file as follows:
   - <error>—If an <error> action is encountered, the mgd fails the commit process, returns an error message to the CLI or JUNOScript client, and takes no further action.

- <warning>—If a <warning> action is encountered, the mgd forwards the message to the CLI or the JUNOScript client.
- <syslog>—If a <syslog> action is encountered, a message is forwarded to the syslog.

4. The mgd processes the action file for <change> actions.

   If any <change> actions are emitted by any commit scripts, the requested changes are loaded into the candidate configuration.

5. The candidate configuration is copied to become the checkout configuration.
6. The mgd processes the action file for <transient-change> actions.

   If any <transient-change> actions are emitted by any commit scripts, the requested changes are loaded into the checkout configuration.

7. The mgd validates the checkout configuration.
8. If no error occurs, the checkout configuration is copied as the current configuration.

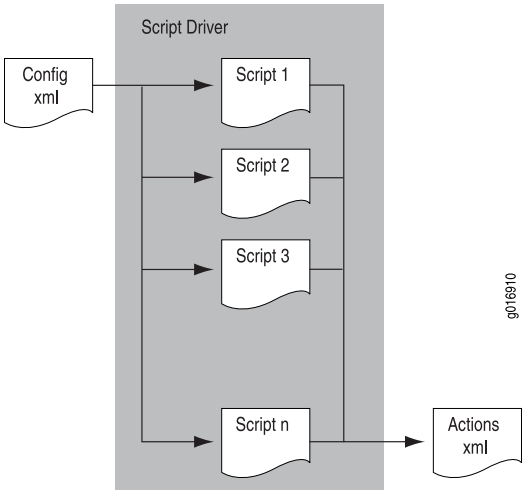
Changes generated by commit scripts are not evaluated by your custom rules the first time they are committed. However, persistent changes are carried in the candidate configuration. During a subsequent commit operation, this process will ensure that the candidate configuration, including past persistent changes, conforms to your custom rules. For more information about candidate changes by commit scripts, see "Using Multiple Commit Scripts" on page 87.

Transient changes are never tested by and do not need to conform to commit script rules. Commit scripts evaluate the candidate configuration, and transient changes are not copied to the candidate configuration. To remove a transient change from the configuration, remove or disable the commit script, or comment out the code that causes the transient change. These concepts are summarized in Table 8 on page 131.

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 evaluation order for commit scripts is based on the order that the commit scripts are listed at the [edit system scripts commit] hierarchy level. See Figure 6 on page 88.

Figure 6: Configuration Evaluation by Multiple Commit Scripts



For example, the commit script A.xsl was added to a router to check its configuration. Among other configuration checks, A.xsl ensures that the domain name server (DNS) is set to 192.168.0.255. Later, this DNS server's address was changed to 192.168.255.255 and a second script, B.xsl, was added to all servers to check that the DNS server was changed to the new address.

When both of these two scripts are run, depending on the original value of the candidate configuration, either script A.xsl or script B.xsl is invoked. If the original candidate configuration lists the incorrect DNS server address of 192.168.0.255, script B.xsl will update the value to 192.168.255.255. However, if the original candidate configuration includes the correct address of 192.168.255.255, script A.xsl will update the value to 192.168.0.255.

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

Using 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 [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 process.

Directly accessing the configuration data and processing non-XML converted data are processor-intensive process 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:

1. Add the direct-access statement at the [system scripts commit] hierarchy.

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

2. Commit the configuration.

Using Commit Scripts

To use commit scripts, follow these steps:

1. Write a commit script.

   This manual provides procedural steps and examples for writing scripts that generate error, warning, or system log messages and configuration changes. For detailed examples, see "Commit Script Examples" on page 189.

2. Copy the script to the /var/db/scripts/commit directory on the hard drive or the /config/scripts/commit directory on the CompactFlash drive, depending on which memory location you wish to use. Only users in the superuser JUNOS login class can access and edit files in the /var/db/scripts/commit or /config/scripts/commit directories.

   If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/commit or /config/scripts/commit directories on each Routing Engine.

NOTE: The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

3. Enable the script by including the file statement at the [edit system scripts commit] hierarchy level. Only users in the superuser class can configure commit scripts.

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

4. Issue the commit command.

The JUNOS software provides several tools to manage and monitor commit script operation, including source control, traceoptions, and monitoring commands. For more information, see "Configuring and Troubleshooting Commit Scripts" on page 163.

Chapter 9

Introduction to Writing Commit Scripts

When you write commit scripts, you can use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided by the JUNOS software. These tools include basic boilerplate that you must include in all commit scripts and an import file called junos.xsl, which includes several extension functions and named templates that make commit scripts easier to read and write.

Commit scripts are based on JUNOScript and JUNOS XML tag elements. Like all XML elements, angle brackets enclose the name of a JUNOScript or JUNOS XML 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 Juniper Networks documentation to indicate optional parts of CLI command strings.

This chapter includes the following topics:

- Boilerplate for Commit Scripts on page 92
- Importing the junos.xsl File on page 94
- Design Considerations on page 103
- Examples: Commit Scripts on page 104

Boilerplate for Commit Scripts

This section contains basic XSLT and SLAX script boilerplate for commit scripts. You must include either XSLT or SLAX boilerplate as the starting point for all commit scripts that you create. The following example shows the XSLT boilerplate:

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">
           <!- - ... insert your code here ... - ->
9      </xsl:template>

10 </xsl:stylesheet>
```

The following example shows the corresponding SLAX code:

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 {
   /*
      * Insert your code here.
   */
}
```

The following is a line-by-line description of the XSLT boilerplate:

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

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.

```
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">
```

Lines 2 through 6 set the stylesheet 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 them all prevents errors if the namespace mappings are used in later versions of the script.

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

Line 7 is an XSLT import statement. It loads the templates and variables from the file **../import/junos.xsl**, which ships as part of the JUNOS software . The **junos.xsl** file contains a set of named templates you can call in your scripts. These named templates are discussed in “Importing the junos.xsl File” on page 94.

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

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 hierarchy level. For more information, see “XPath” on page 17.

The commit script code is added between lines 8 and 9.

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

Line 9 closes the template.

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

Line 10 closes the style sheet and the commit script.

Importing the junos.xsl File

The import file junos.xsl contains several useful templates that you can call within a commit script. To use these templates, you must map to the jcs namespace in your style sheet declaration. You must also import the junos.xsl file. Both of these steps are shown below:

```
<?xml version=”1.0”?>
  <xsl:stylesheet verion=”1.0”
mlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
  <xsl:import href="../import/junos.xsl"/>
  ...
</xsl: stylesheet>
```

Both the namespace mapping and the <xsl:import> element are contained in the basic script boilerplate presented in “Boilerplate for Commit Scripts” on page 92.

For more information, see the following sections:

- Extension Functions in the junos.xsl File on page 94
- Templates in the junos.xsl File on page 96

Extension Functions in the junos.xsl File

The import file junos.xsl contains the following extension functions:

- jcs:invoke()
- jcs:progress()
- jcs:output()
- jcs:trace()
- jcs:first-of()
- jcs:printf()
- jcs:sleep()

Extension functions in the junos.xsl file use the jcs prefix to avoid conflicting with standard XSLT functions. The functions in the jcs namespace allow you to accomplish scripting tasks more easily. To use these functions in your scripts, you simply include variable declarations, a variable call with the select="jcs:*function*()" attribute, and pass along any required or optional arguments.

These functions are discussed in more detail in the following sections:

- jcs:invoke() Function on page 95
- jcs:progress() Function on page 95
- jcs:output() Function on page 95
- jcs:trace() Function on page 95
- jcs:first-of() Function on page 95

- jcs:printf() Function on page 96
- jcs:sleep() Function on page 96

jcs:invoke() Function

The jcs:invoke() function invokes an RPC. It can be called with one argument, either a string containing a JUNOS XML or JUNOScript RPC method name or a tree containing an RPC. The result is the contents of the <rpc-reply> element, not the <rpc-reply> tag element itself.

In the following example, there is a test to see if the interface argument is included on the command line when the script is executed. If it is, the operational mode output of the show interfaces terse command is narrowed to include information about that interface only.

```
<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 an RPC without modifying the output:

```
<xsl:variable name="sw" select="jcs:invoke('get-software-information')"/>
```

jcs:progress() Function

The jcs:progress() function issues a progress message. It sends a progress message back to the client (CLI) containing its single argument.

```
<xsl:variable name="ignore" select="jcs:progress('Working...')"/>
```

jcs:output() Function

The jcs:output() function allows you to make unformatted output text. It emits an <output> element. The text appears in the CLI.

jcs:trace() Function

This function issues a trace message, which is sent to the trace file.

jcs:first-of() Function

This function returns the first nonempty (non-null) item in a list. 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")
```

The following example selects the description of a logical interface if there is a logical interface description. If not, it selects the description of the (parent) physical interface if there is a physical interface description. If not, it selects the concatenation of the physical interface name with a " ." and the logical unit number.

```
<xsl:variable name="description"
      select="jcs:first-of(description, ../description, concat(../name, '.', name))"/>
```

jcs:printf() Function

This function emits formatted output text. The text appears in the CLI. Most standard printf formats are supported, in addition to some JUNOS software-specific ones. The %j1 operator emits the field only if the field was changed from the last time the function was run. The %jc operator capitalizes the first letter of the format output. The %jt{TAG} operator emits the tag if the field is not empty.

```
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ –> }s\n',
                        'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/>
```

jcs:sleep() Function

This function causes the script to sleep for a specified number of seconds and optionally, milliseconds. You can use this function to help determine how a routing component is working over time. To do this, write a script that issues a command, calls the jcs:sleep() function, and reissues the same command.

The *milliseconds* argument is optional. For example, jcs:sleep(1) means 1 second and jcs:sleep(0, 10) means 10 milliseconds.

```
jcs:sleep(seconds <, milliseconds>)
```

Templates in the junos.xsl File

The import file junos.xsl contains the following templates:

- <jcs:edit-path>
- <jcs:emit-change>
- <jcs:emit-comment>
- <jcs:statement>
- <xsl:template match="/">

Named templates in the junos.xsl file use the jcs prefix to avoid conflicting with templates defined in commit scripts. The templates in the jcs namespace allow you to accomplish scripting tasks more easily. To use these templates in your scripts, you simply include <xsl:call-template name="*name*"> elements and pass in any

required or optional parameters. The name attribute specifies the name of the called template.

The <xsl:template match="/"> template is an unnamed template that allows you to use shortened XPath expressions in your scripts.

These templates are discussed in more detail in the following sections:

- < jcs:edit-path > Template on page 97
- < jcs:emit-change > Template on page 97
- < jcs:emit-comment > Template on page 100
- < jcs:statement > Template on page 100
- < xsl:template match = "/" > Template on page 101

<jcs:edit-path> Template

This template generates an <edit-path> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The location of the configuration error is passed as dot into the <jcs:edit-path> template. This location defaults to " ." , the current position in the XML hierarchy. You can alter the default by including the select attribute of the dot parameter. The following example demonstrates how to call this 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 <xnm:warning> element results in the following command-line interface (CLI) output:

```
[edit]
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

This template generates a <change> element, which results in a persistent change to the configuration.

This template includes the following optional parameters:

- <xsl:param name="content">—Allows you to include the content of the change, relative to dot.
- <xsl:param name="dot" select=".">—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current

context "." as a default value. To change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.

- <xsl:param name="message">—Allows you to include a warning message 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, include the dot parameter.
- <xsl:param name="name" select="name($dot)"/>—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 name of the element in $dot (which in turn defaults to ".", the current element).
- <xsl:param name="tag" select="'change'"/>—Allows you to specify the type of change to be generated. By default, the <jcs:emit-change> template generates a permanent change, as designated by the 'change' expression. To specify a transient change, you must include the tag parameter and include the 'transient-change' expression, as shown here:

```
<xsl:with-param name="tag" select="'transient-change'"/>
```

The following example demonstrates how to call this 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 Intermediate System-to-Intermediate System (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 you want the change to affect a node other than the current context node, you can change the context by setting the "dot" parameter to another node. In the following example, the current context node is the [edit interfaces *interface-name* unit] hierarchy level. The "dot" parameter changes the context node 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"/>
         ...
```

<jcs:emit-comment> Template

This template emits 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. 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:statement> Template

This template generates a <statement> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The parameter dot can be passed into the <jcs:statement> template if the error is not at the current position in the XML hierarchy. The following example demonstrates how to call this 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:

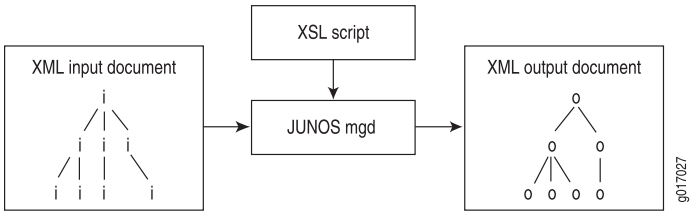
```
[edit]
user@host# commit
[edit interfaces interface so-1/2/3]
        'mtu 576;'  # mtu statement generated by the <jcs:statement> template
     SONET interfaces must have a minimum MTU of 2048.
error: 1 error reported by commit scripts
error: commit script failure
```

The test of the MTU setting is not performed in the <xnm:error> element. For the full example, see "Imposing a Minimum MTU Setting" on page 195.

<xsl:template match="/"> Template

The JUNOS management process (mgd) generates the output document as the product of its evaluation of the input document, as shown in Figure 7 on page 101.

Figure 7: 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 software'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-output>
3           <xsl:apply-templates select="commit-script-input/configuration"/>
4       </commit-script-output>
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-output>.

```
2 <commit-script-output>
```

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.

The following example contains the <xsl:if> programming instruction and the <xnm:warning> element. The logical result of both templates is:

```
<commit-script-output>    <!- - 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 router.</message>
      </xnm:warning>
   </xsl:if>    <!- - end of "configuration" template - ->
</commit-script-output>    <!- - 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 a router 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')">
```

Design Considerations

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

XSLT is an interpreted language, making performance an ever-present issue. Tasks should be done with minimal node traversals and minimal 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 interfaces (cstm1-):

```
<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 "Limiting the Number of E1 Interfaces" on page 199.)

```
[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
```

Examples: Commit Scripts

The following examples illustrate how commit scripts work. Each example is followed by a line-by-line explanation.

First Example

The following script applies a transient change to each interface whose name begins with so-, setting the encapsulation to ppp. (For a SLAX version of this example, see "SLAX Syntax" on page 212.)

```
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>
```

Explanation of First Example

Following is an explanation of each line in the script.

Lines 1 through 8 are the commit script boilerplate. They are explained in "Boilerplate for Commit Scripts" on page 92.

```
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">
```

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 unit.

```
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 JUNOScript <load-configuration> operation.

```
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>
```

Second Example The following sample script finds interfaces that have an International Organization for Standardization (ISO) protocol enabled, and ensures that these interfaces also have Multiprotocol Label Switching (MPLS) enabled and are included at the [edit protocols mpls interface] hierarchy level. (For a SLAX version of this example, see "SLAX Syntax" on page 212.)

```
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>
231                           </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>
```

Explanation of Second Example

Following is an explanation of each line in the script. For brevity, Lines 1 through 7 of the template boilerplate are omitted.

Line 8 opens the template and is explained in "Boilerplate for Commit Scripts" on page 92.

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

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 the JUNOS software, 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 discussed in "Importing the junos.xsl File" on page 94.

```
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>
```

Chapter 10

Generating a Custom Warning, Error, or System Log Message

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 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 CLI or a JUNOS XML or JUNOScript 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 “Generating a Persistent or Transient Configuration Change” on page 129.

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 CLI or to a JUNOS XML or JUNOScript 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.

The JUNOS software generates system log messages (also called syslog messages) to record events that occur on the routing platform, including the following:

- Routine operations, such as creation of an Open Shortest Path First (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 routing platform power-down due to excessive temperature

Each system log message identifies the JUNOS software process that generated the message and briefly describes the operation or error that occurred. The *JUNOS 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.

This chapter discusses the following topics:

- Generating a Custom Warning, Error, or System Log Message on page 110
- Examples: Generating a Custom Warning, Error, or System Log Message on page 113
- Message Tags on page 120

Generating a Custom Warning, Error, or System Log Message

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

1. Include the following Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) template boilerplate.

   You must include either the XSLT or SLAX boilerplate in all commit scripts. For detailed information about this boilerplate, see "Boilerplate for Commit Scripts" on page 92.

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. Include one or more XSLT programming instructions or equivalent SLAX instructions. A few common XSLT constructs are shown here. (For detailed

information, see “Summary of XPath and XSLT Functions, Elements, Attributes, and Templates” on page 27 and “Summary of SLAX Statements” on page 63.)

- <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 “Programming Instructions” on page 22. For information about writing your scripts in SLAX instead of XSLT, see “Understanding SLAX” on page 51.

3. Include the <xnm:warning>, <xnm:error>, or <syslog> element with the <message> child element. You use the <message> child element to specify 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 “Generating a Persistent or Transient Configuration Change” on page 129.)

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 “Message Tags” on page 120.

For SLAX versions of these constructs, see “SLAX Syntax” on page 114, “SLAX Syntax” on page 116, and “SLAX Syntax” on page 119.

4. After you are finished with your commit script, save the script with the name *filename*.
5. Depending on whether you load your scripts from your flash drive or your hard drive, copy the script to he /config/scripts/commit or the /var/db/scripts/commit directory on your routing platform. For more information on commit scripts and the memory location, see "Storing the Commit Scripts" on page 82

   If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/commit or the /config/scripts/commit directory on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.
6. Enable the script by including the file statement at the [edit system scripts commit] hierarchy level:

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

Examples: Generating a Custom Warning, Error, or System Log Message

This section is organized as follows:

- Example: Generating a Custom Warning Message on page 113
- Example: Generating a Custom Error Message on page 115
- Example: Generating a Custom System Log Message on page 118

Example: Generating a Custom Warning Message

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 in Step 3.

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.";
      }
   }
}
```

Verifying the Commit Script Warning Output

To test the output of a warning message, 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 chapter, perform the following steps:

1. Copy the XSLT script shown just previously into a text file, and name the file source-route.xsl.
2. Depending on whether you save your commit script files to the flash drive or the hard drive, copy the source-route.xsl file to the /config/scripts/commit or the /var/db/scripts/commit directory on your routing platform.
3. Include the file source-route.xsl statement at the [edit system scripts commit] hierarchy level:

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

4. 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
   ```

5. Issue the commit command.

When you issue the commit command, the following output appears:

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

To display the Extensible Markup Language (XML) output of a 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/7.4R1/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
2005-06-14 14:40:29 PDT: reading commit script configuration
2005-06-14 14:40:29 PDT: testing commit script configuration
2005-06-14 14:40:29 PDT: opening commit script
'/var/db/scripts/commit/source-route-warning.xsl'
2005-06-14 14:40:29 PDT: reading commit script 'source-route-warning.xsl'
2005-06-14 14:40:29 PDT: running commit script 'source-route-warning.xsl'
2005-06-14 14:40:29 PDT: processing commit script 'source-route-warning.xsl'
[edit chassis]
    warning: IP source-route processing is not enabled.
2005-06-14 14:40:29 PDT: no errors from source-route-warning.xsl
2005-06-14 14:40:29 PDT: saving commit script changes
2005-06-14 14:40:29 PDT: summary: changes 0, transients 0 (allowed), syslog 0
2005-06-14 14:40:29 PDT: no commit script changes
2005-06-14 14:40:29 PDT: exporting juniper.conf
2005-06-14 14:40:29 PDT: expanding groups
2005-06-14 14:40:29 PDT: finished expanding groups
2005-06-14 14:40:29 PDT: setup foreign files
2005-06-14 14:40:29 PDT: propagating foreign files
2005-06-14 14:40:30 PDT: complete foreign files
2005-06-14 14:40:30 PDT: daemons checking new configuration
configuration check succeeds
```

Example: Generating a Custom Error Message

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:

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.";
            }
        }
    }
}
```

Verifying the Commit Script Error Output

To test the output of an error message, make sure that the candidate configuration contains the condition that elicits the error. For this example, ensure that a T1 interface is configured but without the description statement.

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

1. Copy the XSLT script shown just previously into a text file, and name the file description.xsl.
2. Depending on whether you save your commit script files to the flash drive or the hard drive, copy the description.xsl file to the config/scripts/commit or the /var/db/scripts/commit directory on your routing platform.
3. Include the file description.xsl statement at the [edit system scripts commit] hierarchy level:

```
user@host> edit
```

```
[edit]

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

4. If there is no T1 interface configured without 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
```

5. Issue the commit command. The following output appears:

```
[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 output of an 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/7.4R1/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
2005-06-14 15:56:09 PDT: reading commit script configuration
2005-06-14 15:56:09 PDT: testing commit script configuration
2005-06-14 15:56:09 PDT: opening commit script '/var/db/scripts/commit/error.xsl'
2005-06-14 15:56:09 PDT: reading commit script 'error.xsl'
2005-06-14 15:56:09 PDT: running commit script 'error.xsl'
2005-06-14 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.
2005-06-14 15:56:09 PDT: 2 errors from script 'error.xsl'
error: 2 errors reported by commit scripts
error: commit script failure
```

Example: Generating a Custom System Log Message

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:

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.";
      }
    }
  }
}
```

Verifying the Commit Script Syslog Output

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 and commit check | display detail configuration mode commands to verify the output of system log messages.

To test the output of a system log message, make sure that the candidate configuration contains the condition that elicits the system log message. In 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 chapter, perform the following steps:

1. Copy the XSLT script into a text file, and name the file read-write.xsl. Depending on whether you save your commit script files to the flash drive or the hard drive, copy the read-write.xsl file to the config/scripts/commit or the /var/db/scripts/commit directory on your routing platform.
2. Include the file read-write.xsl statement at the [edit system scripts commit] hierarchy level:

```
user@host> edit

[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. Ensure that system logging is configured by issuing the following configuration mode command:

```
[edit]

user@host# show system syslog
```

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

5. Issue the commit command:

```
[edit]

user@host# commit
```

After you issue the commit command, check the log file. The default directory for log files is /var/log. A common filename for the log file is messages. Commit script system log entries look like this:

```
timestamp router-id cscript: message
```

For example:

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

Message Tags

Table 7 on page 121 describes the data that you can include in a custom warning message. To see how data values are supplied within a script, see "Examples: Generating a Custom Warning, Error, or System Log Message" on page 113 and the sample scripts in "Commit Script Examples" on page 189. (For detailed information about element hierarchy, see "Summary of Message Tag Elements" on page 123.)

Table 7: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages

Data Item, XML Element, or Attribute	Required or Supported	Description
Container Tags		
<xnm:error>	Required for error messages	Indicates that the server has encountered a problem while processing the client application's request.
<syslog>	Required for system log messages	Indicates that a system log message is going to be recorded.
<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 7: 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 software 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. 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. 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.

Chapter 11

Summary of Message Tag Elements

This chapter lists the tags that you can use when you create custom warning, error, and system log (syslog) messages. The tag names are in alphabetical order.

<syslog>

Usage

```
<syslog="namespace-URL"  xmlns:xnm="namespace-URL">
  <message>syslog-message </message>
</syslog>
```

Release Information Statement introduced in JUNOS Release 7.4.

Description Record events that occur on the routing platform.

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 "Generating a Custom Warning, Error, or System Log Message" on page 109

<xnm:error>

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 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 software 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 "Generating a Custom Warning, Error, or System Log Message" on page 109

Related Topics < xnm:warning > on page 126

<xnm:warning>

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 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 software 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 "Generating a Custom Warning, Error, or System Log Message" on page 109

Related Topics <xnm:error> on page 124

Chapter 12

Generating a Persistent or Transient Configuration Change

You can use a commit script to specify configuration rules that you always want to enforce. If a rule is broken, then the commit script corrects the problem by changing the candidate configuration with a *persistent* or *transient* configuration change.

A change is persistent in the sense that if you subsequently remove or disable the commit script and then issue a new commit command, the change persists in the configuration and continues to affect routing operations. In short, if the commit script is removed, the persistent change is not removed with it.

A transient change, in contrast, causes a change to be generated in the *checkout configuration* but not in the candidate configuration. The checkout configuration is the configuration database that is inspected for standard JUNOS syntax just before a configuration becomes active. The change is transient in the sense that if you subsequently remove or disable the associated commit script and then issue a new commit command, the change no longer exists in the checkout or active configuration and no longer affects the routing platform components. In short, if the commit script is removed, the transient change is removed with it. You can use transient changes to eliminate the need to repeatedly configure and display well-known policies, thus allowing these policies to be enforced implicitly. For example, if Multiprotocol Label Switching (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.

Table 8 on page 131 describes the differences between persistent and transient configuration 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. This concept is demonstrated later in this chapter.

The <change> and <transient-change> elements are similar to the <load-configuration> element in the JUNOScript API. The possible contents of the <change> and <transient-change> elements are the same as the contents of the <configuration> tag element in the JUNOScript <load-configuration> operation. For complete details about the <load-configuration> element, see the *JUNOS XML API Configuration Reference*.

This chapter discusses the following topics:

- Persistent and Transient Changes on page 130
- Generating a Persistent or Transient Change on page 133
- Examples: Generating a Persistent or Transient Change on page 137
- Removing a Persistent or Transient Change on page 142
- Persistent and Transient Change Tags on page 143

Persistent and Transient Changes

When a candidate configuration includes statements that you have decided must not be included in your configuration, or when the candidate omits statements that you have decided are required, commit scripts can automatically change the candidate and thereby correct the problem. The configuration change can be *persistent*, meaning it is copied to the candidate configuration. Persistent changes remain in the candidate configuration unless you explicitly delete them. Alternatively, the change can be *transient*. Transient changes are loaded into the checkout configuration, but not into the candidate configuration. This means transient changes are not saved in the configuration if the associated commit script is deleted or deactivated.

A persistent or transient change can generate any syntactically correct configuration that you can create by directly editing the configuration using the JUNOS command-line interface (CLI). 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 JUNOS configuration groups, see the *JUNOS System Basics Configuration Guide*.

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 routing platform configuration.

Persistent and transient changes have several important differences, as described in Table 8 on page 131.

Table 8: Differences Between Persistent and Transient Changes

Persistent Changes	Transient Changes
A persistent change is represented in a commit script by the <change> tag. Another way to represent a persistent change is with the content parameter inside a call to the <jcs:emit-change> template. 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. 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 JUNOScript 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 JUNOScript operation. However, some JUNOScript 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 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. Like persistent changes, transient changes must pass the standard JUNOS validity check. 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 merge command in that they cause the software to merge the incoming configuration into the current candidate configuration. If the existing configuration and the persistent change contain conflicting statements, the statements in the persistent change override those in the existing configuration.	Transient changes work like the load update command in that they cause the software to replace only the configuration that has changed. Transient changes are not copied to the candidate configuration. For this reason, transient changes are not saved in the configuration if the associated commit script is deleted or deactivated.
After a persistent change is committed, the software treats it like a change you make by directly editing and committing the candidate configuration. After the persistent changes are copied to the candidate configuration, they are copied to the checkout configuration. If the changes pass the standard JUNOS validity checks, the changes are propagated to the routing platform components.	Each time a transient change is committed, the software updates the checkout configuration database. After the transient changes pass the standard JUNOS validity checks, the changes are propagated to the routing platform components.

Table 8: Differences Between Persistent and Transient Changes *(continued)*

Persistent Changes	Transient Changes
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: user@host# **show** 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: user@host# **show | display commit-scripts** This command displays both persistent and transient changes.
Persistent changes must conform to your custom configuration design rules as dictated by commit scripts. 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 commit model, which is explained in detail in "Commit Scripts and the JUNOS Software Commit Model" on page 85.
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 CLI, because the change is not in the candidate configuration. To alter the contents of a transient change, you must alter the statements in the commit script that generates the transient change.

Generating a Persistent or Transient Change

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

1. Include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) template boilerplate.

   You must include either the XSLT or SLAX boilerplate in all commit scripts. For detailed information about this boilerplate, see "Boilerplate for Commit Scripts" on page 92.

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. After the script element <xsl:template match="configuration">, include one or more XSLT programming instructions:

   - <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 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 "Programming Instructions" on page 22. For information about writing your scripts in SLAX instead of XSLT, see "Understanding SLAX" on page 51.

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 persistent 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" on page 97.)

```
<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 "Generating a Custom Warning, Error, or System Log Message" on page 109.

```
<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" on page 97.)

```
<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. After you are finished with your commit script, save the script with the name *filename*.
5. Copy the script to the /var/db/scripts/commit directory on your routing platform.

   If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/commit directory on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

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

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

7. For transient changes only, 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 persistent changes are loaded into the candidate configuration and then the checkout configuration. Any transient changes are loaded into the checkout configuration only. The commit process then continues with the normal process of validating the configuration and propagating changes to the affected processes on the routing platform.

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

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

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

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

Persistent changes work like the load merge command in that they cause the software to merge the incoming configuration into the current candidate configuration. If the existing configuration and the persistent change contain conflicting statements, the statements in the persistent change override those in the existing configuration. Transient changes work like the load update command in that they cause the software to replace only the configuration that has changed.

Examples: Generating a Persistent or Transient Change

This section is organized as follows:

- Example: Generating a Persistent Change on page 137
- Example: Generating a Transient Change on page 140

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.

The persistent change is generated by the <jcs:emit-change> template, which is a helper template contained in the junos.xsl import file. 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 at the CLI, notifying you that the configuration has been changed.

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>;
               }
            }
         }
      }
   }
}
```

Verifying the Commit Script Output

To test the output of a warning message, make sure that the candidate configuration contains the condition that elicits the warning. For this example, ensure that the family mpls statement is not included at the [edit interfaces so-*fpc/pic/port* unit *logical-unit-number*] hierarchy level.

To display the output of a 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/7.4R1/junos">
    <commit-results>
        <routing-engine junos:style="normal">
            <name>re0</name>
            <xnm:warning>
                <edit-path>
                    [edit interfaces interface so-2/3/4 unit 0]
                </edit-path>
                <message>
                    Adding 'family mpls' to SONET interface.
                </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
2005-06-17 14:17:35 PDT: reading commit script configuration
2005-06-17 14:17:35 PDT: testing commit script configuration
2005-06-17 14:17:35 PDT: opening commit script '/var/db/scripts/commit/mpls.xsl'
2005-06-17 14:17:35 PDT: reading commit script 'mpls.xsl'
2005-06-17 14:17:35 PDT: running commit script 'mpls.xsl'
2005-06-17 14:17:35 PDT: processing commit script 'mpls.xsl'
2005-06-17 14:17:35 PDT: no errors from mpls.xsl
2005-06-17 14:17:35 PDT: saving commit script changes
2005-06-17 14:17:35 PDT: summary: changes 0, transients 0 (allowed), syslog 0
2005-06-17 14:17:35 PDT: no commit script changes
2005-06-17 14:17:35 PDT: finished loading commit script changes
2005-06-17 14:17:35 PDT: exporting juniper.conf
2005-06-17 14:17:35 PDT: expanding groups
2005-06-17 14:17:35 PDT: finished expanding groups
2005-06-17 14:17:35 PDT: setup foreign files
2005-06-17 14:17:35 PDT: propagating foreign files
2005-06-17 14:17:35 PDT: complete foreign files
2005-06-17 14:17:36 PDT: daemons checking new configuration
configuration check succeeds
```

To view the configuration with the persistent change, issue 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, something similar to the following appears after the commit script runs:

```
[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

Using a commit script, make a transient configuration change that sets PPP encapsulation on all SONET/SDH interfaces with the IPv4 protocol family enabled:

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";
         }
      }
   }
}
```

Verifying the Commit Script Output

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

```
[edit]
user@host# commit check | display detail
2005-06-14 12:07:30 PDT: reading commit script configuration
2005-06-14 12:07:30 PDT: testing commit script configuration
2005-06-14 12:07:30 PDT: opening commit script
'/var/db/scripts/commit/transient.xsl'
2005-06-14 12:07:30 PDT: reading commit script 'transient.xsl'
2005-06-14 12:07:30 PDT: running commit script 'transient.xsl'
2005-06-14 12:07:30 PDT: processing commit script 'transient.xsl'
2005-06-14 12:07:30 PDT: no errors from transient.xsl
2005-06-14 12:07:30 PDT: saving commit script changes
2005-06-14 12:07:30 PDT: summary: changes 0, transients 2 (allowed), syslog 0
2005-06-14 12:07:30 PDT: no commit script changes
2005-06-14 12:07:30 PDT: exporting juniper.conf
2005-06-14 12:07:30 PDT: loading transient changes
2005-06-14 12:07:30 PDT: loading commit script changes(transient)
2005-06-14 12:07:30 PDT: finished loading commit script changes
2005-06-14 12:07:30 PDT: expanding groups
2005-06-14 12:07:30 PDT: finished expanding groups
2005-06-14 12:07:30 PDT: setup foreign files
2005-06-14 12:07:30 PDT: propagating foreign files
2005-06-14 12:07:31 PDT: complete foreign files
2005-06-14 12:07:31 PDT: daemons checking new configuration
configuration check succeeds
```

To display the configuration with the transient change, issue the show interfaces | display commit-scripts configuration mode command. If there are one or more SONET/SDH interfaces with the IPv4 protocol family enabled, the output is similar to this:

```
[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 ...
```

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)

   user@host# deactivate (statement | identifier)
   ```

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

   ```
   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

   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 the /var/db/scripts/commit directory on your routing platform. 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
```

Persistent and Transient Change Tags

Table 9 on page 143 describes the data that you can include in a commit script <change> tag. To see how data values are supplied within a script, see "Examples: Generating a Persistent or Transient Change" on page 137 and the sample scripts in "Commit Script Examples" on page 189. (For detailed information about element hierarchy, see "Summary of XSLT Change Tag Elements" on page 159.)

Table 9: Tags and Attributes for Creating Configuration Changes

Data Item, XML Element, or Attribute	Description
Container Tags	
<change>	Request that the JUNOScript server load configuration data into the candidate configuration.
<change [action="*action*"]>	Set the action attribute to merge, override, replace, or update. By default, the action is merge for persistent changes and update for transient changes. The data enclosed in the <change> tag must be JUNOS XML tag elements only.
<change rollback="*index*"/>	Set the rollback attribute to the numerical index of a previous configuration. The routing platform stores a copy of the most recently committed configuration and up to 49 previous configurations. The specified previous configuration completely replaces the current configuration.
<change url="url" [action="*action*"]>	Set the url attribute to the pathname of a file that resides on the routing platform and contains the configuration data to load. The action can be merge, override, replace, or update. By default, the action is merge for persistent changes and update for transient changes. The data must be JUNOS XML tag elements only.

Table 9: Tags and Attributes for Creating Configuration Changes *(continued)*

Data Item, XML Element, or Attribute	Description
<configuration>	Enclose JUNOS XML and JUNOScript tag elements in a <configuration> tag element. You do not explicitly include this tag element in your scripts, because it is implicitly inserted by the script boilerplate. For more information, see “Boilerplate for Commit Scripts” on page 92.
<transient-change>	Request that the JUNOScript 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. 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. You use the tag parameter with the <jcs:emit-change> template. By default, the <jcs:emit-change> template converts the contents of the content parameter into a <change> (persistent change) request.

Chapter 13

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 JUNOS software configuration and automatically correct configuration mistakes when they occur (as discussed in "Generating a Persistent or Transient Configuration Change" on page 129). 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.

This chapter contains an example that shows how macros can simplify a complex Multiprotocol Label Switching (MPLS) configuration. For a demonstration of the time-saving power of macros, see "Automatically Configuring Logical Interfaces and IP Addresses" on page 249.

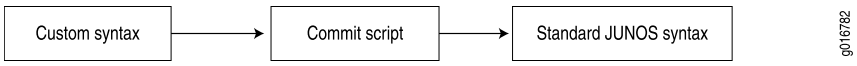
This chapter discusses the following topics:

- How Macros Work on page 145
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 152
- Example: Creating Custom Configuration Syntax with Macros on page 153
- Verifying the Commit Script Output on page 157

How Macros Work

Your custom syntax serves as input to a commit script. The output of the commit script is standard JUNOS configuration syntax, as shown in Figure 8 on page 145. The standard JUNOS 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 configuration statements. If you use transient changes, the custom syntax appears in the candidate configuration, and the standard JUNOS syntax is copied to the checkout configuration only. If you use persistent changes, both the custom syntax and the standard JUNOS syntax appear in the candidate configuration.

This section discusses the following topics:

- Creating a Custom Syntax on page 146
- <data> Element on page 147
- Expanding the Custom Syntax on page 148
- Other Ways to Use Macros on page 151

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 apply-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 segment, 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]
user@host# show
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/7.4R1/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 allows 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']">
```

In SLAX, the syntax looks like this:

```
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 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>
```

In SLAX, the syntax uses the underscore (_) to concatenate values:

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

Now that you have provided the script with instructions to find the necessary data, you can provide content for a transient change that uses the data to construct a standard JUNOS 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 " . " . 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>
```

In SLAX, the syntax looks like this:

```
<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 153.

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

```
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 146 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 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 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 146, you can also use macros to prevent a commit script from performing a task. For example, a basic commit script that automatically adds Multiprotocol Label Switching (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 a configuration example showing this use of macros, see "Controlling LDP Configuration" on page 220.

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 routing platform, but the data can be carried in the configuration file to be used by external applications.

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 routing platform. To give meaning to your syntax, there must be a corresponding commit script that uses the syntax as data for generating related standard JUNOS statements. To write such a script, follow these steps:

1. Include the Extensible Stylesheet Language Transformations (XSLT) or SLAX template boilerplate.

   You must include either the XSLT or SLAX boilerplate in all commit scripts. For detailed information about this boilerplate, see "Boilerplate for Commit Scripts" on page 92.

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. Include one or more XSLT or SLAX programming instructions that inspect the configuration for the apply-macro statement at a specified hierarchy level.
3. Include one or more XSLT or SLAX programming instructions that change the configuration to include standard JUNOS CLI syntax.

   For an XSLT example, see "XSLT Syntax" on page 154. For a SLAX example, see "SLAX Syntax" on page 157.
4. After you are finished with your commit script, save the script with the name *filename*.
5. Copy the script to the /var/db/scripts/commit directory on your routing platform.

   If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/commit

directory on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

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

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

7. If your script includes 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 routing platform.

Example: Creating Custom Configuration Syntax with Macros

Figure 9 on page 153 shows a macro that uses custom syntax and the corresponding expansion to standard JUNOS command-line interface (CLI) syntax.

Figure 9: Sample Macro and Corresponding JUNOS CLI Expansion

Custom macro syntax:

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

macro expansion →

Expanded JUNOS CLI syntax:

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

g017029

In this example, the JUNOS 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 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 153 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.

```
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">
```

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 153 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 153 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, see “Configuring Administrative Groups for LSPs” on page 238.

Verifying the Commit Script Output

When you issue the show protocols mpls | display commit-scripts configuration mode command, the output is similar to the following:

```
[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;
}
```

Chapter 14

Summary of XSLT Change Tag Elements

This chapter lists the XSLT tags that you can use when you create custom permanent or transient changes. The tag names are in alphabetical order.

<change>

Usage

```
<change rollback="index"/>

<change url="url" [action="(merge | override | replace | update)"] />

<change [action="(merge | override | replace | update)"] ]>
 <!- - tag elements representing configuration statements to load - ->
</change>
```

Release Information Statement introduced in JUNOS Release 7.4.

Description Request that the JUNOScript server load configuration data into the candidate configuration. Provide the data to load in one of three ways:

- Set the empty <change/> tag's rollback attribute to the numerical index of a previous configuration. The routing platform stores a copy of the most recently committed configuration and up to 49 previous configurations. The specified previous configuration completely replaces the current configuration.
- Set the empty <change/> tag's url attribute to the pathname of a file that resides on the routing platform and contains the JUNOS XML-encoded configuration data.
- Enclose 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.

Attributes action—Specifies how to load the configuration data, particularly when the candidate configuration and loaded configuration contain conflicting statements. The following are acceptable values:

- merge—Combines the data in the loaded configuration with the candidate configuration. If statements in the loaded configuration conflict with statements in the candidate configuration, the loaded statements replace the candidate ones. This is the default behavior if the action attribute is omitted.
- override—Discards the entire candidate configuration and replaces it with the loaded configuration. When the configuration is later committed, all system processes parse the new configuration.
- replace—Substitutes each hierarchy level or configuration object defined in the loaded configuration that has the replace="replace" attribute for the corresponding level or object in the candidate configuration.

  Also set the replace attribute to the value replace on the opening tag of the container tag element that represents the hierarchy level or object to replace. For more information, see the *JUNOScript API Guide*.
- update—Compares the loaded configuration and candidate configuration. For each hierarchy level or configuration object that is different in the two configurations, the version in the loaded configuration replaces the version in the candidate configuration. When the configuration is later committed, only

system processes that are affected by the changed configuration elements parse the new configuration.

rollback—Specifies the numerical index of the previous configuration to load. Valid values are 0 (zero, for the most recently committed configuration) through one less than the number of stored previous configurations (maximum is 49).

url—Specifies the full pathname of the file that contains the configuration data to load. The file must reside on the routing platform's local disk.

Usage Guidelines "Generating a Persistent or Transient Configuration Change" on page 129 and "Creating Custom Configuration Syntax with Macros" on page 145

Related Topics < transient-change > on page 161

<transient-change>

Usage

```
<transient-change url="url"/>
<transient-change>
 <!- - tag elements representing configuration statements to load - ->
</transient-change>
```

Release Information Statement introduced in JUNOS Release 7.4.

Description Request that the JUNOScript server load configuration data into the checkout configuration. Provide the data to load in one of two ways:

- Set the empty <transient-change/> tag's url attribute to the pathname of a file that resides on the routing platform and contains the JUNOS XMLS-encoded configuration data.

  In the following example, the url attribute identifies /tmp/add.conf as the file to load.

  ```
  <transient-change url="/tmp/add.conf"/>
  ```

- Enclose 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 "Generating a Persistent or Transient Configuration Change" on page 129 and "Creating Custom Configuration Syntax with Macros" on page 145

Related Topics < change > on page 160

Chapter 15

Configuring and Troubleshooting Commit Scripts

At commit time, the JUNOS management process (mgd) looks in the /commit/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. The software runs each commit script against the candidate configuration database to ensure the configuration conforms to the rules dictated by the scripts.

This chapter discusses the following topics:

- Adding and Removing Commit Scripts from the Configuration on page 163
- Using Remote Commit Scripts on page 166
- Manually Converting a Script from XSLT to SLAX on page 169
- Manually Converting a Script from SLAX to XSLT on page 170
- Displaying Commit Script Output on page 170
- Tracing Commit Script Processing on page 172
- Troubleshooting Commit Scripts on page 176

Adding and Removing Commit Scripts from the Configuration

Commit scripts are physically located in the /config/scripts/commit or the /var/db/scripts/commit directories. They are ignored if they are not added to the configuration. However, the system does not recognize these scripts unless the commit script's filename is at the [edit system scripts commit] hierarchy level. To remove a commit script from the configuration, remove the commit script's filename from the [edit system scripts commit] hierarchy level.

Commit Scripts listed at the [edit system scripts commit] hierarchy can include the optional statement. The optional statement, makes the listed commit script optional to the commit process. Adding this statement to the commit script allows a commit operation to succeed even if the script is missing from the commit script directory on the routing platform. For example, you can include the optional statement if you anticipate the need to quickly remove a script from operation by simply deleting it from the commit script directory and you do not want to remove the commit script from the [edit system scripts commit] hierarchy. Later you can replace the file in the commit script directory, and the commit script will once again be active on the router.

Commit scripts can also be deactivated and reactivated as needed. When deactivated, the commit script is listed in the configuration as inactive and ignored during the commit process. Later, when reactivated, the commit script is added back into the commit process.

When written in SLAX, a commit script's filename must include the .slax extension. If the commit script does not include the .slax file extension, the script is not executed. When written in XSLT, a commit script does not require a file extension. However, we strongly recommend that you include the optional .xsl extension to the filename.

CAUTION: If the script you enabled is missing from the commit script directory, you might not discover its absence, since the optional statement allows the commit operation to proceed normally when a file is missing. No error message alerts you that the enabled script is not found.

- Adding Commit Scripts to the Configuration on page 164
- Removing Commit Scripts from the Configuration on page 165
- Deactivating a Commit Script on page 165
- Activating a Commit Script on page 165

Adding Commit Scripts to the Configuration

To add a commit script to the configuration, follow these steps:

1. Ensure that the commit script is located in the correct commit script directory. Commit scripts that are located on the hard drive should be placed in the /var/db/scripts/commit directory. Commit scripts that are located on the CompactFlash drive should be located in the /config/scripts/commit directory. For more information on commit scripts and memory location, see "Storing the Commit Scripts" on page 82.

2. Add the script to the [edit system scripts commit] hierarchy level:

   set system scripts commit file *filename* [optional]

   - file *filename*—The name of the commit script.

     If the commit script is written in XSLT, using the .xsl file extension is optional, but highly recommended. If the commit script is written in SLAX, the file must include the .slax file extension.

   - optional—When this statement is set, the listed commit script is optional to the commit operation. If the commit script is not located within the script directory, the commit operation skips the file and continues with the commit. If this statement is not set, the script is required. If the file is not found within the commit directory, the commit operation fails.

3. Commit your changes:

   commit

Removing Commit Scripts from the Configuration

To remove a commit script from the configuration, follow these steps:

1. Remove the script from the [edit system scripts commit] hierarchy level:

   delete system scripts commit file *filename*

   file *filename*—Name of the commit script.

2. Remove the commit script from the correct 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:

   commit

Deactivating a Commit Script

To deactivate a commit script in the configuration, follow these steps:

1. Issue the deactivate command.

   deactivate system scripts commit file *filename*

   - file *filename*—Name of the commit script:
   - system scripts commit—This statement is issued at the top of the [edit] hierarchy level. These hierarchy levels are omitted if you are in the [edit system scripts commit] hierarchy level.

2. Commit your changes:

   commit

Activating a Commit Script

To activate an inactive commit script in the configuration, follow these steps:

1. Issue the activate command against the commit script:

   activate system scripts commit file *filename*

   file *filename*—Name of the commit script.

2. Commit your changes:

   commit

Using Remote Commit Scripts

You can access and update a router's local commit scripts with a remote copy of these commit scripts. This eases file management in many ways. First, if a specific script is shared by many routers, you can edit and save the script on one router. Then you can access and refresh this updated script on the other routers. You can also create a master commit script location. All scripts can then be maintained at this location. Routers can update their local copies as appropriate by accessing this file location.

For each routing platform, the router continues to use the locally saved commit scripts during a commit operation. The refresh and refresh-from statements are used to update the local commit scripts with the remote commit scripts. The source statement allows you to specify a remote location where the commit scripts are stored.

- Refreshing Commit Script Files on page 166
- Refreshing the Local Commit Script on page 168
- Specifying a Remote Commit Script Source URL on page 169

Refreshing Commit Script Files

You can refresh local commit scripts with remote commit scripts using the refresh and the refresh-from statements. You specify the remote commit script's URL when issuing refresh-from statement. When you issue the refresh statement, the router uses the commit script URL specified by the source statement. For more information on the source statement and defining a master commit script source URL, see "Specifying a Remote Commit Script Source URL" on page 169.

Once the refresh or the refresh-from statement is issued, the router attempts to connect to the remote commit script source and retrieve the remote commit script. If successful, the router updates the local commit script. If a problem occurs, a set of error messages is returned.

The refresh operation occurs as soon as you add the refresh or refresh-from statement to the configuration. In this way, these statements behave like operational mode commands. Further, these statements are not permanently recorded in the configuration file.

If a platform has dual Routing Engines, you need to refresh the commit scripts on both Routing Engines. The commit synchronize command does not refresh the commit scripts between Routing Engines.

CAUTION: We recommend that you do not automate the refresh 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 automating the refresh function for the following reasons:

- Automated refresh means that the network must be in operation in order to successfully commit a configuration. If the network goes down after you make a configuration error, you cannot recover quickly.
- If the software must refresh multiple commit scripts for each commit operation, the network response time can slow down.
- If you automate the refresh operation, the script refresh is the last action in the current commit operation. Consequently, the updated commit script takes effect only for the subsequent 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. For more information, see "Commit Scripts and the JUNOS Software Commit Model" on page 85. In contrast, if you perform a refresh operation manually, the updated commit script takes effect as expected, that is, immediately after your commit the refresh statement in the configuration.
- If you automate the refresh 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 "Refreshing the Local Commit Script" on page 168.

Using the refresh-from Statement on a Single Commit Script

To refresh a single commit script, include the refresh-from statement at the [edit system scripts commit file *filename*] hierarchy level:

```
[edit system scripts commit file filename]
refresh-from url;
```

Specify the source as an HTTP URL, FTP URL, or Secure Copy Protocol (SCP)–style remote file specification.

The refresh operation occurs as soon as you include the refresh-from statement in the configuration and issue the commit command. The refresh-from statement is not carried in the configuration file. In this way, it behaves like an operational mode command.

Using the refresh-from Statement Globally

To refresh all scripts from a specified source, Include the refresh-from statement at the [edit system scripts commit] hierarchy level:

```
[edit system scripts commit]
refresh-from url;
```

- url—URL of the remote commit script directory. The URL can use HTTP, FTP, SCP to access the remote files.

Once the statement has been issued, the router will attempt to connect to the commit script source and retrieve the remote commit script files.

Using the refresh Statement on a Single Commit Script

To refresh a single commit script, issue the refresh statement at the [edit system scripts commit file *filename*] hierarchy level:

```
[edit system scripts commit]
set file filename refresh;
[edit system scripts commit file filename]
source http://my.example.com/pub/scripts/iso.xsl;
```

- file *filename*—The name of the commit script.

  If the commit script is written in XSLT, using the .xsl file extension is optional, but highly recommended. If the commit script is written in SLAX, the file must include the .slax file extension.
- url—URL of the remote commit script directory. The URL can use HTTP, FTP, SCP to access the remote files.

Once the statement has been issued, the router attempts to connect to the commit script source and retrieve the remote commit script file.

Using the refresh Statement Globally

To refresh all scripts from their sources issue the refresh statement at the [edit system scripts commit] hierarchy level:

```
[edit system scripts commit]
set refresh
```

Once the statement has been issued, the router attempts to connect to the commit script source and retrieve all remote commit script files.

Refreshing the Local Commit Script

If the network location of the master source copy defined by the source statement at the [edit system scripts commit file *filename*] hierarchy level becomes unreachable, you can refresh a commit script with a copy from a different location. To do this, include the refresh-from statement at the [edit system scripts commit file *filename*] hierarchy level:

```
[edit system scripts commit file filename]
refresh-from url;
```

Specify the source as an HTTP URL, FTP URL, or SCP-style remote file.

The refresh operation occurs as soon as you include the refresh-from statement in the configuration and issue the commit command. The refresh-from statement is not carried in the configuration file. In this way, it behaves like an operational mode command.

To refresh all scripts from a specified source, include the refresh-from statement at the [edit system scripts commit] hierarchy level:

```
[edit system scripts commit]
refresh-from url;
```

Specify the source as an HTTP URL, FTP URL, or SCP-style remote file.

If a platform has dual Routing Engines and you want the script to be refreshed on both Routing Engines, you must include the refresh-from statement in the configuration of both Routing Engines. The commit synchronize command does not cause the refresh-from statement to take effect on scripts in both Routing Engine directories.

The refresh and refresh-from statements are mutually exclusive.

Specifying a Remote Commit Script Source URL

To indicate a remote source file location for a commit script, follow these steps:

1. Include the source statement at the [edit system scripts commit file *filename*] hierarchy level:

   ```
   [edit system scripts commit]
   set file filename source url;
   ```

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

     If the commit script is written in XSLT, using the .xsl file extension is optional, but highly recommended. If the commit script is written in SLAX, the file must include the .slax file extension.

   - url—The URL of the commit script's master source file. The URL can use the HTTP, FTP, or SCP.

2. Commit your change.

   ```
   [edit system scripts commit]
       file iso.xsl {
           http://my.example.com/pub/scripts/iso.xsl;
       }
   ```

Manually Converting a Script from XSLT to SLAX

Before the JUNOS software invokes the XSLT processor, the software converts SLAX constructs (such as if/then/else) to equivalent XSLT constructs (such as <xsl:choose> and <xsl:if>). For more information about SLAX, see "Understanding SLAX" on page 51.

To convert an XSLT script to SLAX, issue the request system scripts convert xslt-to-slax source *source/filename* destination *destination* operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.xsl file remains in the /var/db/scripts/commit and a new script called script1.slax is added to the /var/db/scripts/commit directory.

To convert a script from SLAX to XSLT, see "Manually Converting a Script from SLAX to XSLT" on page 170.

Manually Converting a Script from SLAX to XSLT

To convert a SLAX script to XSLT, issue the request system scripts convert slax-to-xslt source /var/db/scripts/commit/*filename* destination /var/db/scripts/ operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.slax file remains in the /var/db/scripts/commit and a new script called script1.xsl is added to the /var/db/scripts/commit directory.

To convert a script from XSLT to SLAX, see "Manually Converting a Script from XSLT to SLAX" on page 169.

Displaying Commit Script Output

Table 10 on page 171 summarizes the 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 172.

Table 10: Commit Script Configuration and Operational Mode Commands

Task	Command
Configuration Mode Commands	
Display errors and warnings generated by commit scripts.	commit or commit check
Display detailed information.	commit | display detail
Display the underlying Extensible Markup Language (XML) data.	commit | display xml
Display the postinheritance contents of the configuration database. This view includes transient changes, but does not include changes made in configuration groups.	show | display commit-scripts
Display the postinheritance contents of the configuration database. This view excludes transient changes.	show | display commit-scripts no-transients
Display the postinheritance configuration in XML format. 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 | display commit-scripts view | 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 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 *script-name*

Tracing Commit Script Processing

Commit script tracing operations track all script operations and record them in a log file. The logged error descriptions provide detailed information to help you solve problems faster.

This section discusses the following topics:

- Minimum Configuration for Enabling and Viewing Traceoptions Output on page 172
- Configuring Traceoptions on page 174

Minimum Configuration for Enabling and Viewing Traceoptions Output

If no commit script trace options are configured yet, 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]
   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]
   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 11 on page 173 summarizes useful filtering commands that display selected portions of the cscript.log file.

Table 11: 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 *script-name*

Example: Minimum Configuration for Enabling and Viewing Traceoptions Output

Display the trace output in the 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 Traceoptions

The default operation of commit script trace files is to log important events in a file called cscript.log located in the /var/log directory. When the file cscript.log reaches 128 kilobytes (KB), it is renamed cscript.log.0, then cscript.log.1, and so on, until there are 10 trace files. Then the oldest trace file (cscript.log.9) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

You cannot change the directory (/var/log) in which trace files are located. However, you can customize the 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;
flag {
    all;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

These statements are described in the following sections:

- Configuring the Commit Script Log Filename on page 174
- Configuring the Number and Size of Commit Script Log Files on page 174
- Configuring the Trace Operations on page 175

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 *filename*.0, then *filename*.1, and so on, until there are 10 trace files. Then the oldest trace file (*filename*.9) 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] 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, *filename* is renamed *filename*.0, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename*.0 is renamed *filename*.1 and *filename* is renamed *filename*.0. This process repeats until there are 20 trace files. Then the oldest file (*filename*.19) is overwritten.

The number of files can be from 2 through 1000 files. The file size of each file can be from 10 KB through 1 gigabyte (GB).

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 system scripts commit traceoptions] hierarchy level:

```
[edit system scripts commit traceoptions]
flag {
    all;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

Table 12 on page 175 describes the meaning of the commit script tracing flags.

Table 12: Commit Script Tracing Flags

Flag	Description	Default Setting
all	Trace all operations.	Off
events	Trace important events.	On
input	Trace commit script input data.	Off
offline	Generate data for offline development.	Off
output	Trace commit script output data.	Off
rpc	Trace commit script RPCs.	Off
xslt	Trace the Extensible Stylesheet Language Transformations (XSLT) library.	Off

Troubleshooting Commit Scripts

After you enable a commit script and issue a commit command, the commit script takes effect immediately.

Table 13 on page 176 describes some common problems that might occur.

Table 13: 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: error: could not open commit script: /var/db/scripts/commit/*filename*: No such file or directory	Make sure the file *filename* is in the /var/db/scripts/commit/ directory on your routing platform.
The following error and warning messages appear: error: invalid transient change generated by commit script: *filename* 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. To rectify this problem, take one of the following actions: ■ Remove the code that generates a transient change from the indicated script. ■ Remove the script. ■ Include the allow-transients statement at the [edit system scripts commit] hierarchy level.

Table 13: Troubleshooting Commit Scripts *(continued)*

Problem	Solution
An expected action does not occur. 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. 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 "Boilerplate for Commit Scripts" on page 92.
	3. Make sure that the Extensible Markup Language Path (XPath) expressions in the script contain valid JUNOS command-line interface (CLI) statements expressed as JUNOScript tag elements. 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. If you nest one instruction inside another, the outer instruction changes the context node, so the inner instruction must be relative to the outer. 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. <xsl:for-each select="interfaces/interface[starts-with(name, 't1-')]"> <xsl:if test="not(description)">

Chapter 16

Summary of Commit Script Configuration Statements

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

allow-transients

Syntax allow-transients;

Hierarchy Level [edit system scripts commit]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS 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.

Usage Guidelines See "Generating a Persistent or Transient Change" on page 133 and "Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements" on page 152.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

apply-macro

Syntax

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

Hierarchy Level All hierarchy levels

Release Information Statement introduced in JUNOS Release 7.4.

Description With commit script macros, use custom syntax in your configuration.

Macros work by locating apply-macro statements that you include in the candidate configuration and using the values specified in the apply-macro statement as parameters to a set of instructions (the macro) defined in a commit script. The commit script alters your configuration from one that contains custom syntax into a full configuration containing standard JUNOS 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.

Usage Guidelines See “Creating Custom Configuration Syntax with Macros” on page 145.

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

commit

Syntax

```
commit {
    allow-transients;
    file filename {
        optional;
        refresh;
        refresh-from url;
        source url;
    }
    refresh;
    refresh-from url;
    traceoptions {
        file filename <files number> <size size>;
        flag flag;
    }
}
```

Hierarchy Level [edit system scripts]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS commit scripts, configure commit-time scripting mechanism.

The statements are explained separately.

Usage Guidelines See “Configuring and Troubleshooting Commit Scripts” on page 163.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

direct-access

Syntax

```
commit {
    direct-access;
}
```

Hierarchy Level [edit system scripts commit]

Release Information Statement introduced in JUNOS Release 9.1.

Description Using this statement, you can specify that commit scripts read input configurations directly from the database when inpecting these scripts for errors.

Usage Guidelines

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

file

Syntax

```
file filename {
    optional;
    refresh;
    refresh-from url;
    source url;
}
```

Hierarchy Level [edit system scripts commit]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS commit scripts, enable a commit script that is located in the /var/db/scripts/commit directory.

Options *filename*—The name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing a commit script.

The statements are explained separately.

Usage Guidelines See "Configuring and Troubleshooting Commit Scripts" on page 163.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

optional

Syntax

```
optional;
```

Hierarchy Level [edit system scripts commit file *filename*]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS 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 routing platform. The optional statement allows a commit operation to progress as though the commit script were not enabled in the configuration.

Usage Guidelines See "Adding and Removing Commit Scripts from the Configuration" on page 163.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

refresh

Syntax refresh;

Hierarchy Level [edit system scripts commit],
[edit system scripts commit file *filename*]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS 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.

Usage Guidelines See "Using Remote Commit Scripts" on page 166, and "Refreshing Commit Script Files" on page 166.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh-from, source

refresh-from

Syntax refresh-from *url*;

Hierarchy Level [edit system scripts commit],
[edit system scripts commit file *filename*]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS commit scripts, overwrite the local copy of all enabled commit scripts or a single enabled script located in the /var/db/scripts/commit directory with the copy located at a URL other than the URL specified in the source statement.

Options *url*—The source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

Usage Guidelines See "Refreshing the Local Commit Script" on page 168.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh, source

scripts

Syntax

```
scripts {
   commit {
      allow-transients;
      file filename {
         optional;
         refresh;
         refresh-from url;
         source url;
      }
      refresh;
      refresh-from url;
   }
}
```

Hierarchy Level [edit system]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS commit scripts, configure scripting mechanisms.

The statements are explained separately.

Usage Guidelines See “Configuring and Troubleshooting Commit Scripts” on page 163.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

source

Syntax source *url*;

Hierarchy Level [edit system scripts commit file *filename*]

Release Information Statement introduced in JUNOS Release 7.4.

Description For JUNOS commit scripts, specify the location of the source file for an enabled script located in the /var/db/scripts/commit directory. When you include the refresh statement at the same hierarchy level and commit the configuration, the local copy is overwritten by the version stored at the specified URL.

Options *url*—The source specified as an HTTP URL, FTP URL, or scp-style remote file specification.

Usage Guidelines See "Using Remote Commit Scripts" on page 166 and "Refreshing Commit Script Files" on page 166.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh, refresh-from

traceoptions

Syntax

```
traceoptions {
    file filename <files number> <size size>;
    flag flag;
}
```

Hierarchy Level [edit system scripts commit]

Release Information Statement introduced in JUNOS Release 7.4.

Description Define tracing operations for commit scripts.

Default If you do not include this statement, no commit-script-specific tracing operations are performed.

Options *filename*—Name of the file to receive the output of the tracing operation. All files are placed in the directory /var/log. By default, commit script process tracing output is placed in the file cscript.log. If you include the file statement, you must specify a filename. To retain the default, you can specify cscript.log as the filename.

files *number*—(Optional) Maximum number of trace files. When a trace file named *trace-file* reaches its maximum size, it is renamed *trace-file*.0, then *trace-file*.1, and so on, until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum number of files, you also must specify a maximum file size with the size option and a filename.
Range: 2 through 1000
Default: 10 files

flag—Tracing operation to perform. To specify more than one tracing operation, include multiple flag statements. You can include the following flags:

- all—Log all operations
- events—Log important events
- input—Log commit script input data
- offline—Generate data for offline development
- output—Log commit script output data
- rpc—Log commit script RPCs
- xslt—Log the XSLT library

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 *trace-file*.0. When *trace-file* again reaches its maximum size, *trace-file*.0

is renamed *trace-file*.1 and *trace-file* is renamed *trace-file*.0. 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: *x*k to specify KB, *x*m to specify MB, or *x*g to specify GB
Range: 10 KB through 1 GB
Default: 128 KB

Usage Guidelines See "Tracing Commit Script Processing" on page 172.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Chapter 17

Commit Script Examples

This chapter includes the following examples:

- Requiring and Restricting Configuration Statements on page 189
- Requiring Internal Clocking on T1 Interfaces on page 192
- Imposing a Minimum MTU Setting on page 195
- Warning About a Deprecated Value on page 197
- Limiting the Number of E1 Interfaces on page 199
- Limiting the Number of ATM Virtual Circuits on page 208
- Controlling IS-IS and MPLS Interfaces on page 211
- Adding T1 Interfaces to a RIP Group on page 214
- Adding a Default Encapsulation Type on page 217
- Controlling LDP Configuration on page 220
- Adding a Final then accept Term to a Firewall on page 224
- Configuring an Interior Gateway Protocol on an Interface on page 228
- Creating a Complex Configuration Based on a Simple Interface Configuration on page 232
- Configuring Administrative Groups for LSPs on page 238
- Controlling a Dual Routing Engine Configuration on page 242
- Preventing Import of the Full Routing Table on page 246
- Automatically Configuring Logical Interfaces and IP Addresses on page 249
- Prepending a Global Policy on page 255
- Assigning a Classifier on page 260
- Loading a Base Configuration on page 263

Requiring and Restricting Configuration Statements

This example shows you how to use commit scripts to specify required and prohibited configuration statements.

This 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 emits an error message and the commit operation fails.

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;
            }
        }
    }
}
```

Testing ex-no-nukes.xsl

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

1. From "Requiring and Restricting Configuration Statements" on page 189, copy the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) script into a text file, and name the file ex-no-nukes.xsl. Copy the ex-no-nukes.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-no-nukes.xsl;
        }
    }
}
interfaces {
    fxp0 {
        disable;
        unit 0 {
            family inet {
                address 10.0.0.1/24;
            }
        }
    }
}
```

3. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command. When you issue the **commit** command, the following output appears:

```
[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
```

Requiring Internal Clocking on T1 Interfaces

This example shows you how to use a commit script to require T1 interfaces to be configured with internal clocking.

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 emitted and the configuration is not committed.

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.";
         }
      }
   }
}
```

Testing ex-clocking-error.xsl

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

1. From "Requiring Internal Clocking on T1 Interfaces" on page 192, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-clocking-error.xsl. Copy the ex-clocking-error.xsl file to the /var/db/scripts/commit directory on your routing platform.

2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-clocking-error.xsl;
        }
    }
}
interfaces {
    t1-0/0/0 {
        clocking external;
    }
    t1-0/0/1 {
        unit 0;
    }
}
```

3. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[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
```

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.

This example tests the MTU of SONET/SDH interfaces, reports when the MTU is less than the value of the $min-mtu variable, here set to 2048, and causes the commit operation to fail.

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 ".";
```

```
        }
      }
    }
  }
```

Testing ex-so-mtu.xsl

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

1. From "Imposing a Minimum MTU Setting" on page 195, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-so-mtu.xsl. Copy the ex-so-mtu.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-so-mtu.xsl;
        }
    }
}
interfaces {
    so-1/2/2 {
        mtu 2048;
    }
    so-1/2/3 {
        mtu 576;
    }
}
```

3. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[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
```

Warning About a Deprecated Value

In previous versions of the JUNOS software, you could include the **speed** statement at the [edit system ports console] hierarchy level with a speed setting lower than 9600 baud. Since Release 7.4, setting the **speed** lower than 9600 baud has been deprecated.

For general information about deprecated configuration statements, see the *JUNOS Release Notes*.

This example allows you to set the **speed** statement at the [edit system ports console] hierarchy level, but the commit script produces an error message if you set the speed lower than 9600 baud.

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="system/ports/console/undocumented/speed &lt; 9600">
         <xnm:warning>
            <xsl:call-template name="jcs:edit-path">
               <xsl:with-param name="dot" select="system/ports/console"/>
            </xsl:call-template>
            <xsl:call-template name="jcs:statement">
               <xsl:with-param name="dot"
                        select="system/ports/console/undocumented/speed"/>
            </xsl:call-template>
            <message>
               <xsl:text>Console speeds less than</xsl:text>
               <xsl:text>9600 baud are deprecated.</xsl:text>
            </message>
         </xnm:warning>
         <change>
            <system>
               <ports>
                  <console>
                     <speed delete="delete"/>
                  </console>
               </ports>
            </system>
         </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 {
   if (system/ports/console/undocumented/speed < 9600) {
      <xnm:warning> {
         call jcs:edit-path($dot = system/ports/console);
         call jcs:statement($dot = system/ports/console/undocumented/speed);
         <message> {
            expr "Console speeds less than";
            expr "9600 baud are deprecated.";
         }
      }
      <change> {
         <system> {
            <ports> {
               <console> {
                  speed delete="delete">;
               }
            }
         }
      }
   }
}
```

Testing ex-deprecated.xsl

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

1. From "Warning About a Deprecated Value" on page 197, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-deprecated.xsl. Copy the ex-deprecated.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

   ```
   system {
      ports {
         console speed 4800;
      }
      scripts {
         commit {
            file ex-deprecated.xsl;
         }
      }
   }
   ```

3. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

   ```
   [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 using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl + d.

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[edit]
user@host# commit
[edit system ports console]
    'speed 4800;'
    warning: Console speeds less than 9600 baud are deprecated.
commit complete
```

Limiting the Number of E1 Interfaces

This example limits the number of E1 interfaces configured on a Channelized STM1 Intelligent Queuing (IQ) Physical Interface Card (PIC).

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 variable $limit. If there are more E1 interfaces than $limit, a commit error is generated and the commit operation fails.

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.";
                }
            }
        }
    }
}
```

Testing ex-16-e1-limit.xsl

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

1. From "Limiting the Number of E1 Interfaces" on page 199, copy the XSLT script into a text file, and name the file ex-16-e1-limit.xsl. Copy the ex-16-e1-limit.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-16-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;
        lmi {
            lmi-type ansi;
        }
        e1-options {
            framing g704;
        }
        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;
```

```
        lmi {
            lmi-type ansi;
        }
        e1-options {
            framing g704;
        }
        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;
        lmi {
            lmi-type ansi;
        }
        e1-options {
            framing g704;
        }
        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;
        lmi {
            lmi-type ansi;
        }
        e1-options {
            framing g704;
        }
        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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
```

```
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
    lmi {
        lmi-type ansi;
    }
    e1-options {
        framing g704;
    }
    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;
```

```
            lmi {
                lmi-type ansi;
            }
            e1-options {
                framing g704;
            }
            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;
            lmi {
                lmi-type ansi;
            }
            e1-options {
                framing g704;
            }
            unit 100 {
                point-to-point;
                dlci 100;
                family inet {
                    address 10.0.0.34/31;
                }
            }
        }
    }
```

3. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command. When you issue the **commit** command, the following output appears:

```
[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
```

Limiting the Number of ATM Virtual Circuits

This example limits the number of Asynchronous Transfer Mode (ATM) virtual circuits (VCs) configured on an ATM interface.

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.

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.";
                }
            }
        }
    }
}
```

Testing ex-atm-vc-limit.xsl

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

1. From "Limiting the Number of ATM Virtual Circuits" on page 208, copy the XSLT script into a text file, and name the file ex-atm-vc-limit.xsl. Copy the ex-atm-vc-limit.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl + d.

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[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
```

Controlling IS-IS and MPLS Interfaces

If you want to enable Multiprotocol Label Switching (MPLS) on an interface, you must make changes at both the [edit interfaces] and [edit protocols mpls] hierarchy levels. This example shows you 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.

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;
                    }
                }
            }
        }
```

```
    }
}
```

Testing ex-iso.xsl

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

1. From "Controlling IS-IS and MPLS Interfaces" on page 211, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-iso.xsl. Copy the ex-iso.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

b. Press Enter.

c. Press Ctrl + d.

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[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
```

5. Issue the show interfaces command. Confirm that the loopback interface is not altered, and 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;
   }
}
```

Adding T1 Interfaces to a RIP Group

If you want to enable the Routing Information Protocol (RIP) on an interface, you must make changes at both the [edit interfaces] and [edit protocols rip] hierarchy levels. This example shows you how to use commit scripts to decrease the amount of manual configuration.

This example adds 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.

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;
                     }
                  }
               }
            }
         }
      }
```

```
    }
}
```

Testing ex-rip-t1.xsl

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

1. From "Adding T1 Interfaces to a RIP Group" on page 214, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-rip-t1.xsl. Copy the ex-rip-t1.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command. When you issue the show protocols rip group test configuration mode command, the following output appears:

```
[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;
```

Adding a Default Encapsulation Type

Point-to-Point Protocol (PPP) encapsulation is the default encapsulation type for physical interfaces. You need not 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.

Regular configuration groups cannot test whether the configuration for an interface at the [edit interfaces *interface-name* sonet-options] hierarchy level includes the aggregate statement. A commit script can perform this test and set the encapsulation only on nonaggregated interfaces.

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";
         }
      }
   }
}
```

Testing ex-so-encap.xsl

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

1. From "Adding a Default Encapsulation Type" on page 217, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-so-encap.xsl. Copy the ex-so-encap.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
```

```
    scripts {
        commit {
            file ex-so-encap.xsl;
            allow-transients;
        }
    }
}
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. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command.

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

Display Configuration with Transient Changes

When you issue the **show interfaces | display commit-scripts** command, the following output appears:

```
[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 taking effect on 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;
        }
    }
}
```

Display Configuration Without Transient Changes

When you issue the show interfaces command, the following output appears. The transient changes are not displayed when you do not include the piped display commit-scripts option. They are in effect nonetheless.

```
[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;
        }
    }
}
```

Controlling LDP Configuration

If you want to enable the Label Distribution Protocol (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 you 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 routing platform, there is no problem; otherwise, a warning is emitted 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:

```
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 emitted.

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 emitted.

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";
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";
                    }
                }
            }
        }
    }
}
```

Testing ex-ldp.xsl

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

1. From "Controlling LDP Configuration" on page 220, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-ldp.xsl. Copy the ex-ldp.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command. When you issue the commit command, the following output appears:

```
[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
```

Adding a Final then accept Term to a Firewall

Each firewall filter in the JUNOS software 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. This example adds a final then accept action to any firewall filter that does not already end with it.

In this example, the commit script adds a then accept statement to any firewall filter that does not already end with an explicit then accept statement.

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>;
               }
```

```
            }
        }
    }
}
}
```

Testing ex-add-accept.xsl

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

1. From "Adding a Final then accept Term to a Firewall" on page 224, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-add-accept.xsl. Copy the ex-add-accept.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command. 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
```

5. Issue the **show firewall** command. 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 */
        }
     }
```

Configuring an Interior Gateway Protocol on an Interface

When you add a new interface to an Open Shortest Path First (OSPF) or Intermediate System-to-Intermediate System (IS-IS) domain, you must configure the interface at multiple hierarchy levels, including [edit interfaces] and [edit protocols]. This example uses a macro to automatically include the interface at the [edit protocols] hierarchy level and to configure the proper interior gateway protocol (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 management process (mgd) 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.

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;
                        }
                    }
                }
            }
        }
    }
}
```

Testing ex-if-class.xsl

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

1. From "Configuring an Interior Gateway Protocol on an Interface" on page 228, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-if-class.xsl. Copy the ex-if-class.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command:

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

Script-Generated Configuration

When you issue the show protocols configuration mode command, the following output appears:

```
[edit]
user@host# show protocols
isis {
    interface t3-0/0/0.0;
}
ospf {
    area 10.4.0.0 {
        interface so-1/2/3.0;
    }
}
```

Manual Configuration

When you issue the show interfaces configuration mode command, the following output appears:

```
[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;
        }
    }
}
```

Creating a Complex Configuration Based on a Simple Interface Configuration

This example uses a 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 management process (mgd) 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, the script does the following:

- 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 a 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.

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;
                    }
                }
            }
        }
    }
}
```

Testing ex-if-params.xsl

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

1. From "Creating a Complex Configuration Based on a Simple Interface Configuration" on page 232, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-if-params.xsl. Copy the ex-if-params.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            allow-transients;
```

```
                file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command:

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

When you issue the show interfaces | display commit-scripts | display inheritance configuration mode command, the following output appears:

```
[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;
        }
    }
}
```

When you issue the show protocols | display commit-scripts configuration mode command, the following output appears:

```
[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;
}
```

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.

In this example, the JUNOS 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 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 the necessary configuration statements, see "Testing ex-lsp-admin.xsl" on page 240.

For a line-by-line explanation of this script, see "Example: Creating Custom Configuration Syntax with Macros" on page 153.

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;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Testing ex-lsp-admin.xsl

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

1. From "Configuring Administrative Groups for LSPs" on page 238, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-lsp-admin.xsl. Copy the ex-lsp-admin.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            allow-transients;
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command:

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

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;
}
```

Controlling a Dual Routing Engine Configuration

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

The JUNOS software 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, ex-dual-re.xsl, emits 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, ex-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 emits 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.

XSLT Syntax: ex-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: ex-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: ex-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: ex-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]";
            }
        }
    }
}
```

Testing ex-dual-re.xsl and ex-dual-re2.xsl

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

1. From "Controlling a Dual Routing Engine Configuration" on page 242, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into two text files. Name one file ex-dual-re.xsl, and name the other file ex-dual-re2.xsl. Copy the ex-dual-re.xsl and ex-dual-re2.xsl files to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.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 ex-dual-re.xsl;
            file ex-dual-re2.xsl;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0
            family inet {
                address 192.168.220.1/30;
            }
        }
    }
}
```

3. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command. When you issue the **commit** command, the following output appears. After the commit operation is completed, the router hostname changes 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
[edit]
user@router1-RE0#
```

Preventing Import of the Full Routing Table

In JUNOS software 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.

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).

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'";
            }
        }
    }
}
```

Testing ex-import.xsl

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

1. From "Preventing Import of the Full Routing Table" on page 246, copy the Extensible Stylesheet Language Transformations (XSLT) or SLAX script into a text file, and name the file ex-import.xsl. Copy the ex-import.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-import.xsl;
        }
    }
}
protocols {
    ospf {
        import bad-news;
    }
}
policy-options {
    policy-statement bad-news {
        then accept;
    }
}
```

3. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. When you issue the commit command, the following output appears:

```
[edit]
user@host# commit
[edit protocols ospf import]
   'import bad-news;'
   policy contains bare 'then accept'
error: 1 error reported by commit scripts
error: commit script failure
```

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 macro to complete the task quickly and with no errors.

This example 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.

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;
}
```

Testing ex-atm-logical.xsl

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

1. From "Automatically Configuring Logical Interfaces and IP Addresses" on page 249, copy the XSLT script into a text file, and name the file ex-atm-logical.xsl. Copy the ex-atm-logical.xsl file to the /var/db/scripts/commit directory on your routing platform.

2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            allow-transients;
            file ex-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. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command:

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

When you issue the show interfaces at-1/2/3 | display commit-scripts configuration mode command, the following output appears:

```
[edit]
user@host# show interfaces at-1/2/3 | display commit-scripts
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;
    }
}
```

Prepending a Global Policy

For most configuration objects, the order in which the object or its children are created is not significant, because the JUNOS 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 ensures that a Border Gateway Protocol (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 emitted, and the commit operation fails.

Otherwise, the commit script uses the insert="before" JUNOScript 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.

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";
                }
            }
        }
    }
}
```

Testing ex-bgp-global-import.xsl

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

1. From "Prepending a Global Policy" on page 255, copy the XSLT script into a text file, and name the file ex-bgp-global-import.xsl. Copy the ex-bgp-global-import.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            allow-transients;
            file ex-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. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command:

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

show protocols

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:

```
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;
        }
    }
}
```

show protocols | display commit-scripts

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:

```
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;
        }
    }
}
```

show protocols | display commit-scripts

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:

```
user@host# set protocols bgp group fish neighbor 192.168.16.6 import green
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;
        }
    }
}
```

Assigning a Classifier

In JUNOS 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.

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";
                        }
                    }
                }
            }
        }
    }
}
```

Testing ex-classifier.xsl

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

1. From "Assigning a Classifier" on page 260, copy the XSLT script into a text file, and name the file ex-classifier.xsl. Copy the ex-classifier.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
system {
    scripts {
        commit {
            file ex-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. Merge the configuration into your routing platform configuration by issuing the **load merge terminal** configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the **commit** command:

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

When you issue the **commit** command, the following output appears:

```
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
```

show class-of-service

When you issue the **show class-of-service** configuration mode command, the **fc-q3** classifier is assigned to the **fe-0/0/0.0** interface:

```
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
        }
```

```
    }
}
```

Loading a Base Configuration

This script is a macro that sets up a router with a sample base configuration. With minimal manual user input, the script automatically configures:

- A router 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 router ID as the loopback address

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";
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/AHkmTQFCA0BIcIKWL7sgaRh-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;
                               }
                            }
                         }
                      }
                   }
                }
             }
          }
       }
    }
```

Testing config-system.xsl

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

1. From "Loading a Base Configuration" on page 263, copy the XSLT script into a text file, and name the file ex-classifier.xsl. Copy the ex-classifier.xsl file to the /var/db/scripts/commit directory on your routing platform.
2. Select the following configuration, and press Ctrl + c to copy it to the clipboard. If you are using the SLAX version of the script, change the filename to *filename*.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 to your router.

3. Merge the configuration into your routing platform configuration by issuing the load merge terminal configuration mode command:

```
[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 using the mouse and the paste icon.

   b. Press Enter.

   c. Press Ctrl + d.

4. Issue the commit command:

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

show | display commit-scripts

When you issue the show | display commit-scripts configuration mode command, the router base configuration should be set up:

```
user@host# show | display commit-scripts
...
```

Part 3
Operation (Op) Scripts

- Operation (op) Scripts Overview on page 281
- Introduction to Writing Operation Scripts on page 285
- Configuring Operation Scripts on page 303
- Operation Script Examples on page 315
- Summary of Op Script Configuration Statements on page 335

Chapter 18

Operation (op) Scripts Overview

JUNOS op scripts automate network and router management and troubleshooting. Op scripts can perform any function available through the remote procedure calls (RPCs) supported by either of the two application programming interfaces (APIs): the JUNOS Extensible Markup Language (XML) API and the JUNOScript API.

Op scripts allow you to do the following things:

- Monitor the overall status of a routing platform.
- Customize the output of operational mode commands.
- Reconfigure the routing platform to avoid or work around known problems in the JUNOS software.
- Change the router's configuration in response to a problem.

This chapter includes the following topics:

- Op Script Programming on page 281
- Operation (op) Scripts on page 282
- Storing Op Scripts on page 282
- How Op Scripts Work on page 282
- Using Operation Scripts on page 283

Op Script Programming

Op scripts are based on two APIs: the JUNOS XML API and the JUNOScript API. For more information on the JUNOS XML API and the JUNOScript API, see "Introduction to the JUNOS XML and JUNOScript APIs" on page 9. 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. For more information about XPath and XSLT, see "Understanding XSLT" on page 15. For more information about SLAX, see "Understanding SLAX" on page 51.

Operation (op) Scripts

Op scripts—Execute operational commands and inspect router output through the command-line interface (CLI). Op scripts are executed by the mgd process

Storing Op Scripts

By default, commit, operation, and event scripts are stored on the router's hard drive. However, you can save these scripts to the flash memory by including the load-scripts-from-flash statement at the [edit system scripts] hierarchy level:

```
[edit system scripts]
load-scripts-from-flash;
```

The load-scripts-from-flash statement applies to all commit, operation, and event scripts, and changes the physical location of these scripts. This statement does not affect script operation. Op scripts located on the router's hard drive are accessed from the /var/db/scripts/op directory. Op scripts located on the router's CompactFlash drive are accessed from the /config/scripts/op directory.

You can view the currently active op scripts on the router in the /var/run/scripts/op directory. This directory allows you to view the active scripts without having to search for this information within the current configuration.

NOTE: When you switch from storing the files on the hard drive to storing files on the flash drive, or you switch from storing the files on the flash drive to storing files on the hard drive, you must manually move any scripts residing in the former memory location to the new memory location. If you do not do this, the scripts are not available to the router.

How Op Scripts Work

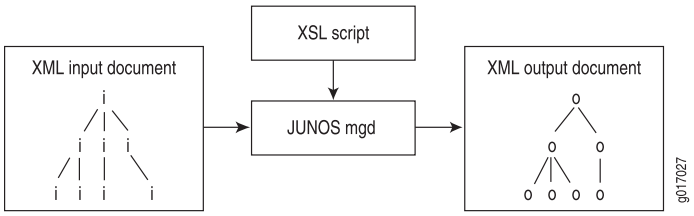
Op scripts execute router operational commands and inspect the resulting output. After inspection, op scripts can automatically correct errors within the router based on this output.

You add op scripts to router 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 on op script file directories, see "Storing Op Scripts" on page 282. Once added to the router, op scripts are invoked from the command line, using the op *filename* command.

You can use op scripts to generate changes to the router 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 routing platform configuration.

Figure 10 on page 283 shows a high-level view of the flow of op script input and output.

Figure 10: Op Script Input and Output



Using Operation Scripts

This section provides directions for using operation scripts on your router.

- Installing Op Scripts on a Router on page 283
- Executing Op Scripts on page 284

Installing Op Scripts on a Router

To install op scripts on a router, follow these steps:

1. Write an op script.

   For information on writing op scripts, see "Introduction to Writing Operation Scripts" on page 285. For examples of op scripts, see "Operation Script Examples" on page 315.

2. Copy the script to the /var/db/scripts/op directory on the hard drive or the /config/scripts/op directory on the flash drive, depending on which memory location is used. Only users in the superuser JUNOS login class can access and edit files in the /var/db/scripts/op and config/scripts/op directories.

NOTE: If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/op or config/scripts/op directories on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

3. Add the script to the [edit system scripts op] hierarchy level using the file statement. Only users in the superuser class can configure op scripts.

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

4. Issue the commit command.

   Once the configuration is committed, the op script will be available on the router. It can be executed using the op command.

Executing Op Scripts

Once an op script is available on the router, you can execute the script in the CLI:

1. Open the CLI in operational mode.
2. Issue the op command:

   user@host> **op *filename***

   filename—The name of the op script you are executing.

The JUNOS software provides several tools to manage and monitor op script operations, including source control and tracing operations. For more information, see “Configuring Operation Scripts” on page 303.

Chapter 19

Introduction to Writing Operation Scripts

When you write operation scripts, you can use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided by the JUNOS software. These tools include basic boilerplate that you must include in all operation scripts and an import file called junos.xsl, which includes several named templates and extension functions that make operation scripts easier to read and write.

Operation scripts are based on JUNOScript and JUNOS XML tag elements. Like all XML elements, angle brackets enclose the name of a *JUNOScript* or JUNOS XML 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 Juniper Networks documentation to indicate optional parts of CLI command strings.

This chapter includes the following topics:

- Boilerplate for Op Scripts on page 285
- Displaying Operational Mode Fields in XML on page 287
- Using RPCs and Operational Mode Commands on page 287
- Importing the junos.xsl File on page 289

Boilerplate for Op Scripts

This section contains basic XSLT and SLAX script boilerplate for op scripts. You must include either XSLT or SLAX boilerplate as the starting point for all op scripts that you create. The XSLT boilerplate is followed by line-by-line comments.

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="/">
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 them all prevents errors if the namespace 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 ../import/junos.xsl, which ships as part of the JUNOS software (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 "Importing the junos.xsl File" on page 289.

```
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 JUNOScript Remote Procedure Calls (RPCs). For more information, see "XPath" on page 17.

```
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>
```

After Line 11, you can define additional XSLT templates that are called from within the <xsl:template match="/"> template.

```
  <!- - ... insert additional template definitions here ... - ->
```

Line 12 closes the style sheet and the op script.

```
12  </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 / {
   <op-script-results> {
      /*
         * Insert your code here.
      */
   }
}
```

Displaying Operational Mode Fields in XML

To write op scripts, you must gather information about the JUNOS XML version of operational mode commands and output fields. You can do this by consulting the *JUNOS XML API Operational Reference*.

Another very useful tool is 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/7.6R1/junos">
    <interface-information
xmlns="http://xml.juniper.net/junos/7.6I0/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

Most operational mode commands supported in the JUNOS software have XML equivalents known as *remote procedure calls* (RPCs). All operational mode commands that have XML equivalents are listed in the *JUNOS XML API Operational Reference*.

Using RPCs

To use an RPC in an op script, include the RPC in a variable declaration, as shown in the following snippet. This snippet is expanded and fully described in "Customizing Output of the show interfaces terse Command" on page 320.

```
<xsl:variable name="rpc">
   <get-interface-information/> # JUNOS RPC for the show interfaces command
</xsl:variable>
<xsl:variable name="out" select="jcs:invoke($rpc)"/>
...
```

Using Operational Mode Commands

Some operational mode commands do not currently have XML equivalents (RPCs). 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 with the | display xml piped command:

```
user@host> operational-mode-command | display xml
```

If the output of this command contains no more detail than the <output> and <cli> tag elements, the command might not have an XML equivalent. For example, the show host command has no XML equivalent. The only child elements of the <rpc-reply> element are the <output> and <cli> tag elements:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/7.6R1/junos">
  <output>
     ...
   </output>
   <cli>
      <banner></banner>
   </cli>
</rpc-reply>
```

NOTE: For some commands, if there is no corresponding configuration, the output of the piped | display xml command might contain no information, even though an RPC exists for the command. For example, suppose a router configuration contains no statements at the [edit class-of-service] hierarchy level. If you issue the show services cos statistics forwarding-class | display xml command on this router, the output does not contain the existing <service-cos-forwarding-class-statistics> response tag:

```
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, checking the *JUNOS XML API Operational Reference* is the most reliable method for determining whether a command has an XML equivalent.

You can reference commands that have no XML equivalent. You can do this by including the <command>, <xsl:value-of>, and <output> elements, as shown in the following snippet. This snippet is expanded and fully described in "Displaying DNS Hostname Information" on page 317.

```
<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>
...
```

Importing the junos.xsl File

The import file junos.xsl contains several useful extension functions that you can call from an operation script. To use these functions, you must include two instructions in your script: the jcs namespace mapping and the <xsl:import> element:

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

Both the namespace mapping and the <xsl:import> element are contained in the basic script boilerplate presented in “Boilerplate for Op Scripts” on page 285.

For more information, see the following sections:

- Extension Functions in the junos.xsl File on page 289
- Templates in the junos.xsl File on page 298

Extension Functions in the junos.xsl File

Extension functions in the junos.xsl file use the jcs prefix to avoid conflicting with standard XSLT functions. The functions in the jcs namespace allow you to accomplish scripting tasks more easily. To use these functions in your scripts, you simply include a *variable declaration*, a variable call with the select="jcs:*function*()" attribute, and pass in any required or optional arguments.

These functions are discussed in more detail in the following sections:

- jcs:break-lines() Function on page 290
- jcs:close() Function on page 290
- jcs:dampen() Function on page 290
- jcs:empty() Function on page 291
- jcs:execute() Function on page 291
- jcs:first-of() Function on page 292
- jcs:getsecret() Function on page 292
- jcs:hostname() Function on page 293
- jcs:input() Function on page 293

- jcs:invoke() Function on page 293
- jcs:open() Function on page 294
- jcs:output() Function on page 294
- jcs:parse-ip() Function on page 294
- jcs:printf() Function on page 295
- jcs:progress() Function on page 295
- jcs:regex() Function on page 295
- jcs:sleep() Function on page 296
- jcs:split() Function on page 296
- jcs:sysctl() Function on page 296
- jcs:syslog() Function on page 297
- jcs:trace() Function on page 298

jcs:break-lines() Function

The jcs:break-lines() function is used to break a single element into multiple elements, delimited by <newline> characters. This function is especially useful in dealing with large output elements, such as those returned using the show pfe command.

The function syntax is as follows:

```
var $lines jcs:break-lines($output);
```

Where:

- $lines—New broken up output.
- $output—Original output.

jcs:close() Function

The jcs:close() function closes a previously opened connection handle.

The function syntax is as follows:

```
expr jcs:close($connection);
```

Where $connection is a connection handle generated by a previously executed jcs:open() function.

jcs:dampen() Function

The jcs:dampen() function is used to limit the number of times a function is called in succession when using a script. If a specific function exceeds the maximum number of calls within a specified amount of time, the jcs:dampen() function returns false. If a function does not exceed the maximum number of calls within a specified amount of time, the function returns true.

The function syntax is as follows:

```
var $rc jcs:dampen($tag, max, frequency);
```

Where:

- $rc—Returned value based on the number of calls within a specified amount of time. If the number of calls exceeds the maximum number allowed, the value is false. If the number of calls is less than the maximum number allowed, the value is true.
- $tag—The function call measured.
- max—The maximum number of function calls allowed. This maximum number is based on the number of calls within a specified time interval (frequency).
- frequency—The time interval, in minutes,

The following example shows how the dampen() function can be used:

```
if (jcs:dampen('my-change', 3, 10)) {
    /* Code to handle situations where the my-change function */
    /*  is called more than 3 times within 10 minutes */
} else {
    /* Code to handle all other cases */
```

jcs:empty() Function

The jcs:empty() function returns true if a nodeset or string argument is empty.

The function syntax is as follows:

```
jcs:empty($set);
```

Where $set is the nodeset or string to test.

The following example shows how the jcs:empty() function can be used:

```
if (jcs:empty($set)); {
    /* Code to handle true value ($set is empty) */
}
```

jcs:execute() Function

The jcs:execute() function executes an RPC within the context of a specified connection handle. Any number of RPCs can be executed within the context of a specified connection handle until that connection handle is closed with the jcs:close() function.

The function syntax is as follows:

```
var $results = jcs:execute($connection, $rpc);
```

Where:

- $results—Results of the executed RPC. This $results varable is the same as the $results variable produced by the jcs:invoke() function.
- $connection—Connection handle generated by a previously executed jcs:open() function.
- $rpc—RPC to be executed. This $rpc varable is the same as the $rpc variable produced by the jcs:invoke() function.

jcs:first-of() Function

The jcs:first-of() function returns the first nonempty (non-null) item in a list. 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")
```

The following example selects the description of a logical interface if there is a logical interface description. If not, it selects the description of the (parent) physical interface if there is a physical interface description. If not, it selects the concatenation of the physical interface name with a " ." and the logical unit number.

```
<xsl:variable name="description"
      select="jcs:first-of(description, ../description, concat(../name, '.', name))"/>
```

jcs:getsecret() Function

The jcs:getsecret() function invokes a CLI prompt and waits for user input similar to the jcs:input() function. However, unlike jcs:input(), the user input will not be prompted/echoed back to the user, which is useful for entering user passwords. The user input is defined as a string for subsequent use. This function cannot be used with event scripts.

The function syntax is as follows

```
var $answer = jcs:getsecret("prompt ")
```

Where:

- prompt—CLI prompt.
- $answer—Returned user input.

The following example shows how to prompt for a password which will not be echoed back to the user:

```
var $password = jcs:getsecret("Enter password: ")
```

jcs:hostname() Function

The jcs:hostname() function returns the hostname associated with a given IP address.

The function syntax is as follows:

```
var $name = jcs:hostname($address);
```

Where:

- $name—Returned hostname.
- $address—IPv4 or IPv6 address.

jcs:input() Function

The jcs:input() function invokes a CLI prompt and waits for user input. The user input is defined as string for subsequent use. This function cannot be used with event scripts.

The function syntax is as follows

```
jcs:input("prompt" $answer)
```

Where:

- prompt—CLI prompt.
- $answer—Returned answer.

jcs:invoke() Function

The jcs:invoke() function invokes an RPC. It can be called with one argument, either a string containing a JUNOS XML or JUNOScript RPC method name or a tree containing an RPC. The result is the contents of the <rpc-reply> element, not the <rpc-reply> tag element itself.

In the following example, there is a test to see if the interface argument is included on the command line when the script is executed. If it is, the operational mode output of the show interfaces terse command is narrowed to include information about that interface only.

```
<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 an RPC without modifying the output:

```
<xsl:variable name="sw" select="jcs:invoke('get-software-information')"/>
```

jcs:open() Function

The jcs:open() function returns a connection handle that can be used to execute multiple RPCs using the jcs:execute() function and close the connection using the jcs:close() function.

The syntax is as follows:

```
var $connection = jcs:open($server, $username, $passphrase)
```

Where:

- $connection—Connection handle returned.
- $server—Domain name or IP address of the remote router. If you are opening a local connection, do not pass the $server value.
- $username—User's login name.
- $passphrase—User's login password.

The following example shows how to connect to a local device:

```
$connection = jcs:open()
```

The following example shows how to connect to a remote device:

```
$connection = jcs:open(remote-router)
```

The following example shows how the user bsmith with a passphrase password can open a connection with the server fivestar:

```
$connection = jcs:open("fivestar", "bsmith", "password")
```

jcs:output() Function

The jcs:output() function allows you to make unformatted output text. It emits an <output> element. The text appears in the CLI.

jcs:parse-ip() Function

The jcs:parse-ip() function evaluates an IPv4 or IPv6 IP address and returns the following information:

- Hostname.
- Address family (inet4 for IPv4 or inet6 for IPv6).
- Prefix length.

- Network address.
- Netmask (for IPv4 address; left blank for IPv6 addresses).

The function syntax is as follows:

```
var $output = jcs:parse-ip(ipaddress/(prefix-length | netmask));
```

Where:

- $output—Results of the executed RPC.
- ipaddress—IPv4 or IPv6 address.
- prefix-length—Prefix length.
- netmask—Netmask.

jcs:printf() Function

The jcs:print() function emits formatted output text. The text appears in the CLI. Most standard printf formats are supported, in addition to some JUNOS software-specific ones. The %j1 operator emits the field only if the field was changed from the last time the function was run. The %jc operator capitalizes the first letter of the format output. The %jt{TAG} operator emits the tag if the field is not empty.

```
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ –> }s\n',
                        'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/>
```

jcs:progress() Function

The jcs:progress() function issues a progress message. It sends a progress message back to the client (CLI) containing its single argument.

```
<xsl:variable name="ignore" select="jcs:progress('Working...')"/>
```

jcs:regex() Function

The jcs:regex() function evaluates a regular expression against a given argument and returns any matches.

The function syntax is as follows:

```
var $pat = "(evaluation-pattern)";
var $output = ($pat, "expression")
```

Where:

- $pat—Specified pattern string to evaluate against the regular expression.
- $output—Matching values.

For example:

```
var $pat = "([0–9=) (:*)([a-z}*)";
```

```
var $a = jcs:regex($pat, "123:xyz");
```

Returns:

```
"123:xyz"
    ([0–9]+) returns "123"
    (:*) returns ":"
    ([a-z]*) returns "xyz"
```

For example:

```
var $pat = "([0–9=) (:*)([a-z}*)";
var $a = jcs:regex($pat, "r2d2");
```

Returns:

```
"2d"
    ([0–9]+) returns "2"
    (:*) returns empty match
    ([a-z]*) returns "d"
```

jcs:sleep() Function

The jcs:sleep() function causes the script to sleep for a specified period. You can use this function to help determine how a routing component is working over time. To do this, write a script that issues a command, calls the jcs:sleep() function, and reissues the same command.

The *millis* argument is optional. For example, jcs:sleep(1) means 1 second and jcs:sleep(0, 10) means 10 milliseconds.

```
jcs:sleep(seconds <, millis>)
```

jcs:split() Function

The jcs:split() function splits a string into an array of substrings based on the regular expression pattern. If the optional argument limit is specified, only substrings up to the limit are returned.

```
jcs:split($pattern, $string, [$limit])
```

jcs:sysctl() Function

The jcs:sysctl() function returns a sysctl value as a string or integer.

The function syntax is as follows:

```
var $value = jcs:sysctl("sysctl-value", "s");
```

Where:

- $value—Returned string or integer value.
- sysctl-value—sysctl value to convert to a string or integer.

jcs:syslog() Function

The jcs:syslog() function logs messages to the syslog with a specified priority.

The function syntax is as follows:

```
expr jcs:syslog(priority, "message", <$goesto>, <$syslog>));
```

Where:

- priority—The priority given to the syslog message. The priority can be specified as a facility/severity pair, or it can be a numeric priority value based on the facility/severity pair. Table 14 on page 297 and Table 15 on page 298 show the facility and severity strings available and the corresponding numeric value for each string.

  The priority numeric value is calculated by multiplying the facility string value by 8 and adding the severity string. For example, if the facility/severity string pair is pfe/alert, the priority numeric value is 161 ((20 x 8) + 1).
- message—Message to add to the system log file.
- $goesto—(Optional) You can pass variable names as an argument to the function.
- $syslog—(Optional) You can pass the name of the system log file as an argument to the function.

The message added to the syslog file is specified within the [edit system syslog] hierarchy level of the router configuration file.

Table 14: 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
firewall	Firewall filtering system	19
ftp	FTP processes	11
interact	Commands executed by thye UI	23
pfe	Packet forwarding engine	20
user	User processes	1

Table 15: Severity String

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

jcs:trace() Function

The jcs:trace() function issues a trace message, which is sent to the trace file.

Templates in the junos.xsl File

Named templates in the junos.xsl file use the jcs prefix to avoid conflicting with templates defined in commit scripts. The templates in the jcs namespace allow you to accomplish scripting tasks more easily. To use these templates in your scripts, you simply include <xsl:call-template name="*name*"> elements and pass in any required or optional parameters. The name attribute specifies the name of the called template.

The <xsl:template match="/"> template is an unnamed template that allows you to use shortened XPath expressions in your scripts.

These templates are discussed in more detail in the following sections:

- <jcs:edit-path> Template on page 298
- <jcs:emit-change> Template on page 299
- <jcs:emit-comment> Template on page 301
- <jcs:statement> Template on page 302

<jcs:edit-path> Template

The <jcs:edit> template generates an <edit-path> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The location of the configuration error is passed as dot into the <jcs:edit-path> template. This location defaults to " . " , the current position in the XML hierarchy. You can alter the default by including the select attribute of the dot parameter. The following example demonstrates how to

call this 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 <xnm:warning> element 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

The <jcs:emit-change> template generates a <change> element, which results in a persistent change to the configuration.

This template includes the following optional parameters:

- <xsl:param name="content">—Allows you to include the content of the change, relative to dot.
- <xsl:param name="dot" select=".">—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current context " ." as a default value. If you want to change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.
- <xsl:param name="message">—Allows you to include a warning message to be displayed by 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, include the dot parameter.
- <xsl:param name="name" select="name($dot)"/>—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 name of the element in $dot (which in turn defaults to " ." which is the current element).
- <xsl:param name="tag" select="'change'"/>—Allows you to specify the type of change to be generated. By default, the <jcs:emit-change> template generates a permanent change, as designated by the 'change' expression. To specify a transient change, you must include the tag parameter and include the 'transient-change' expression, as shown here:

  ```
  <xsl:with-param name="tag" select="'transient-change'"/>
  ```

The following example demonstrates how to call this 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 Intermediate System-to-Intermediate System (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:with-param name="dot" select="chassis"/>
```

<jcs:emit-comment> Template

The <jcs:emit-comment> template emits 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. 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:statement> Template

The <jcs:statement> template generates a <statement> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The parameter dot can be passed into the <jcs:statement> template if the error is not at the current position in the XML hierarchy. The following example demonstrates how to call this 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 “Imposing a Minimum MTU Setting” on page 195.

Chapter 20

Configuring Operation Scripts

Operation 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 operation scripts.

To configure operation scripts, include the following statements at the [edit] hierarchy level:

```
system {
    scripts {
        op {
            file filename {
                arguments {
                    argument-name {
                        description descriptive-text;
                    }
                }
                command filename-alias;
                description descriptive-text;
                refresh;
                refresh-from url;
                source url;
            }
            refresh;
            refresh-from url;
            traceoptions {
                file filename <files number> <size size> <world-readable | no-world-readable>;
                flag flag;
            }
        }
    }
}
```

This chapter discusses the following topics:

- Enabling an Operation Script and Defining a Script Alias on page 304
- Executing an Operation Script on page 304
- Declaring Arguments in Op Scripts on page 305
- Configuring Command-Line Help Text on page 307
- Specifying a Master Source for an Op Script on page 308
- Refreshing an Op Script from the Master Source on page 308

- Refreshing an Op Script from a Different Location on page 309
- Manually Converting a Script from XSLT to SLAX on page 310
- Manually Converting a Script from SLAX to XSLT on page 310
- Tracing Op Script Processing on page 311

Enabling an Operation Script and Defining a Script Alias

After you write an operation script, you must save it in the /var/db/scripts/op directory on the routing platform. Operation scripts in the /var/db/scripts/op directories are ignored if they are not enabled. Only users in the superuser JUNOS login class can access and edit files in the /var/db/scripts/op directory.

If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/op directory on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

To enable an operation script, include the file statement at the [edit system scripts op] hierarchy level, specifying the name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an op script. When you enable a SLAX script, you must include the .slax extension in the filename. If you do not, the configuration commits, but the script cannot be executed. When you enable an XSLT script, the .xsl extension is optional but strongly recommended.

For op scripts, the optional command statement allows you to specify an alias for the filename of the script. Thus, you can run the script by invoking either the filename or the filename alias.

```
[edit system scripts op]
file filename {
   command filename-alias;
}
```

Executing an Operation Script

Operation scripts do not take effect when you issue a commit command. This command places the script into the system memory and enables it for execution on the system. Once the script is committed, you then execute the script to run it. Op scripts are executed using the op command on the CLI.

Executing an Op Script

The op script does not take effect when you issue a commit command. Rather, you execute an op script by issuing the op *filename* or op *filename-alias* operational mode command:

```
user@host> op filename

- OR -
```

```
user@host> op filename-alias
```

Declaring Arguments in Op Scripts

There are two ways to declare arguments to an op script: with XSLT (or SLAX) instructions in the script or with JUNOS statements in the 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 Command-Line Help Text" on page 307.

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"/>
```

In SLAX, the syntax looks like this:

```
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 for the argument. 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 do not appear among the Possible completions when you issue the op *filename* ? command:

```
user@host> op filename ?
Possible completions:
...Configuration-generated arguments appear.
...Script-generated arguments are hidden.
```

If you declare all arguments in the script (and none in the configuration), then the arguments do appear in Possible completions. This is because the management process (mgd) populates the Possible completions list by first checking the configuration for arguments. The management 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"/>
```

Method 1: In the script1 Op Script

```
<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>
```

Method 2: In the Configuration

```
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
```

Configuring Command-Line Help Text

For JUNOS op scripts, you can provide a help-text string that appears in the CLI to describe the script and the script arguments. To configure help text, 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 *name*]

The following examples show the configuration and the resulting output.

Examples: Configuring Command-Line Help Text

Configure help text for a filename and for arguments:

For a Filename

```
[edit system scripts op file interface]
user@host# set description "Test the interface" # Configuration
[edit system scripts op]
user@host# set file ?
Possible completions:
<name> Local filename of the script file
interface.xsl Test the interface # Output
```

For Arguments

```
[edit system scripts op file interface arguments t1]
user@host# set description "Search for T1 interfaces" # Configuration
[edit system scripts op file interface arguments t3]
user@host# set description "Search for T3 interfaces" # Configuration
  [edit system scripts op file interface arguments]
```

```
user@host# set ?
Possible completions:
  <name>          Name of the argument
  t1              Search for T1 interfaces # Output
  t3              Search for T3 interfaces
```

Specifying a Master Source for an Op Script

You can store a master copy of each op script in a central repository. This eases file management because you can make changes to the master op script in one place and then refresh the local copies of the currently enabled op scripts.

Specifying a Master Source for an Op Script

To indicate the location of the master source file of an op script, include the source statement at the [edit system scripts op file *filename*] hierarchy level:

```
[edit system scripts op file filename]
source url;
```

Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification. The following example uses an HTTP URL:

```
[edit]
system {
   scripts {
      op {
         file iso {
            source http://my.example.com/pub/scripts/iso.xsl;
         }
      }
   }
}
```

For each routing platform, the local source of the op script remains the /var/db/scripts/op directory on the routing platform. The source statement allows you to refresh the copy located in the /var/db/scripts/op directory by overwriting it with the master copy located at the specified URL.

The source statement has no effect until you include the refresh statement in the configuration, as described in “Refreshing an Op Script from the Master Source” on page 308.

Refreshing an Op Script from the Master Source

To refresh a local op script with a copy from the location specified in the source statement, include the refresh statement at the [edit system scripts op file *filename*] hierarchy level:

```
[edit system scripts op file filename]
refresh;
```

When you commit the configuration and the refresh operation is performed, the copy of the XSLT or SLAX file located in the /var/db/scripts/op directory is overwritten by the copy of the file located at the URL you configure in the source statement at the same hierarchy level. If the script is enabled as described in "Enabling an Operation Script and Defining a Script Alias" on page 304, the refreshed copy of the operation script runs immediately during the current commit operation.

To refresh all scripts from their sources, include the refresh statement at the [edit system scripts op] hierarchy level:

```
[edit system scripts op]
refresh;
```

If a platform has dual Routing Engines and you want the script to be refreshed on both Routing Engines, you must include the refresh statement in the configuration of both Routing Engines. The commit synchronize command does not cause the refresh statement to take effect on scripts in both Routing Engine directories.

The refresh operation happens as soon as you include the refresh statement in the configuration and issue the commit command. The refresh statement is not permanently recorded in the configuration file. In this way, it behaves like an operational mode command.

Refreshing an Op Script from a Different Location

In case the network location of the master source copy defined by the source statement at the [edit system scripts op file *filename*] hierarchy level becomes unreachable, you can refresh an op script with a copy from a different location. To do this, include the refresh-from statement at the [edit system scripts op file *filename*] hierarchy level:

```
[edit system scripts op file filename]
refresh-from url;
```

Specify the source as an HTTP URL, FTP URL, or scp-style remote file specification.

The refresh operation happens as soon as you include the refresh-from statement in the configuration and issue the commit command. The refresh-from statement is not carried in the configuration file. In this way, it behaves like an operational mode command.

To refresh all scripts from a specified source, include the refresh-from statement at the [edit system scripts op] hierarchy level:

```
[edit system scripts op]
refresh-from url;
```

Specify the source as an HTTP URL, FTP URL, or scp-style remote file specification.

If a platform has dual Routing Engines and you want the script to be refreshed on both Routing Engines, you must include the refresh-from statement in the configuration of both Routing Engines. The commit synchronize command does not cause the refresh-from statement to take effect on scripts in both Routing Engine directories.

The refresh and refresh-from statements are mutually exclusive.

Manually Converting a Script from XSLT to SLAX

SLAX is a C-like alternative syntax to XSLT and can be viewed as a preprocessor for XSLT. Before the JUNOS software invokes the XSLT processor, the software converts SLAX constructs (such as if/then/else) to equivalent XSLT constructs (such as <xsl:choose> and <xsl:if>). For more information about SLAX, see "Understanding SLAX" on page 51.

To convert an XSLT script to SLAX, issue the request system scripts convert xslt-to-slax source /var/db/scripts/op/*filename* destination /var/db/scripts/ operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.xsl file remains in the /var/db/scripts/op directory and a new script called script1.slax is added to the /var/db/scripts/op directory.

To convert a script from SLAX to XSLT, see "Manually Converting a Script from SLAX to XSLT" on page 310.

Manually Converting a Script from SLAX to XSLT

To convert a SLAX script to XSLT, issue the request system scripts convert slax-to-xslt source /var/db/scripts/op/*filename* destination /var/db/scripts/ operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.slax file remains in the /var/db/scripts/op directory and a new script called script1.xsl is added to the /var/db/scripts/op directory.

To convert a script from XSLT to SLAX, see "Manually Converting a Script from XSLT to SLAX" on page 310.

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 trace files 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 and saved with a number 0 through 9 in ascending order appended to the end of the file. For example: op-script.log.0, then op-script.log.1, until there are 10 trace files. Then the oldest trace file (op-script.log.9) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

This chapter discusses the following topics:

- Minimum Configuration for Enabling and Viewing Traceoptions Output on page 311
- Configuring Traceoptions on page 312

Minimum Configuration for Enabling and Viewing Traceoptions Output

If no operation script trace options are configured yet, 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]
   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]
   traceoptions flag output;
   ```

3. Issue the commit command:

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

4. Display the resulting trace messages recorded in the /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 16 on page 312 summarizes useful filtering commands that display selected portions of the op-script.log file.

Table 16: Op Script Tracing Operational Mode Commands

Task	Command
Display logging data associated with all script processing.	show log op-script.log
Display script processing for only the most recent commit operation.	show log op-script.log | last
Display processing for script errors.	show log op-script.log | match error
Display script processing for a particular script.	show log op-script.log | match *script-name*

Example: Minimum Configuration for Enabling and Viewing Traceoptions Output

Trace the output of the op script file source-route:

```
system {
    scripts {
        op {
            file source-route;
            traceoptions flag output;
        }
    }
}
[edit]
user@host# commit
[edit]
user@host# run show log op-script.log | last
```

Configuring Traceoptions

You cannot change the /var/logdirectory in which trace files are located. However, you can customize the 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;
flag {
    all;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

These statements are described in the following sections:

- Configuring the Op Script Log Filename on page 313
- Configuring the Number and Size of Op Script Log Files on page 313
- Configuring the Op Script Trace Operations on page 313

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 *filename*.0, then *filename*.1, and so on, until there are 10 trace files. Then the oldest trace file (*filename*.9) 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] or [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, *filename* is renamed *filename*.0, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename*.0 is renamed *filename*.1 and *filename* is renamed *filename*.0. This process repeats until there are 20 trace files. Then the oldest file*filename*.19 is overwritten.

The number of files can be from 2 through 1000 files. The size of each file can be from 10 KB through 1 gigabyte (GB).

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;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

Table 17 on page 314 describes the meaning of the op script tracing flags.

Table 17: 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

Chapter 21

Operation Script Examples

This chapter provides sample op scripts that run commands and customize output. This chapter includes the following topics:

- Restarting an FPC on page 315
- Displaying DNS Hostname Information on page 317
- Customizing Output of the show interfaces terse Command on page 320
- Finding LSPs to Multiple Destinations on page 330
- Importing and Exporting Files on page 333

Restarting an FPC

This example simply restarts a Flexible PIC Concentrator (FPC) and slightly modifies the output of the request chassis fpc command to include the FPC number that is restarting.

There is no JUNOS Extensible Markup Language (XML) equivalent for the request chassis commands. Therefore, this script uses the request chassis fpc command directly rather that using a remote procedure call (RPC). For more information, see "Using RPCs and Operational Mode Commands" on page 287.

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.";
    }
  }
}
```

Testing ex-fpc.xsl

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

1. From "Restarting an FPC" on page 315, copy the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) script into a text file, and name the file ex-fpc.xsl. Copy the ex-fpc.xsl file to the /var/db/scripts/op directory on your routing platform.
2. Include the file ex-fpc.xsl statement at the [edit system scripts op] hierarchy level. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
[edit system scripts op]
```

file ex-fpc.xsl;

3. Issue the commit and-quit command.
4. When you issue the op ex-fpc slot *number* command, the output looks like this:

```
user@host> op ex-fpc slot 0

Restarting the FPC in slot 0. To verify, issue the "show chassis fpc"
command.
```

Displaying DNS Hostname Information

This script displays Domain Name System (DNS) information for a routing platform 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 routing platform 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 that using a remote procedure call (RPC). For more information, see "Using RPCs and Operational Mode Commands" on page 287.

The script is shown as two distinct versions. These two versions accept the same argument and produce the same output. Version 1 uses the <xsl:choose> element. Version 2 uses the jcs:first-of() function.

XSLT Syntax 1: <xsl:choose>

```
<?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 2: jcs:first-of()

```
<?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 1: <xsl:choose>

```
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 2: jcs:first-of()

```
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;
    }
}
```

Testing ex-hostname.xsl

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

1. From "Displaying DNS Hostname Information" on page 317, copy the XSLT or SLAX script into a text file, and name the file ex-hostname.xsl. Copy the ex-hostname.xsl file to the /var/db/scripts/op directory on your routing platform.
2. Include the file ex-hostname.xsl statement at the [edit system scripts op] hierarchy level. If you are using the SLAX version of the script, change the filename to *filename*.slax.

```
[edit system scripts op]
file ex-hostname.xsl;
```

3. Issue the commit and-quit command.
4. When you issue the op ex-hostname operational mode command without the dns argument, the DNS information is displayed for the router you are currently using:

```
[edit]
user@host# op ex-hostname
Name:
this-router has address 10.168.71.246
```

When you issue the op ex-hostname dns *hostname* command, the DNS information is displayed for the specified router:

```
[edit]
user@host# op ex-hostname dns router1
Name:
router1 has address 10.168.71.249
```

When you issue the op ex-hostname dns *address* command, the DNS information is displayed for the specified address:

```
[edit]
user@host# op ex-hostname dns 10.168.71.249
Name:
249.71.168.10.IN-ADDR.ARPA domain name pointer router1
```

Customizing Output of the show interfaces terse Command

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/7.6I0/junos">
  <interface-information xmlns="http://xml.juniper.net/junos/7.6I0/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>
            ...
            (Output truncated)
```

XSLT Syntax The following script customizes the output of the show interfaces terse command.

```
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: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>
18      <xsl:param name="interface"/>
19      <xsl:param name="protocol"/>
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>
32              <xsl:variable name="out" select="jcs:invoke($rpc)"/>
33              <interface-information junos:style="terse">
```

```
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 = $protocol]">
37                          <xsl:call-template name="intf"/>
38                      </xsl:for-each>
39                  </xsl:when>
40                  <xsl:when test="$protocol">
41                      <xnm:error>
42                          <message>
43                              <xsl:text>invalid protocol: </xsl:text>
44                              <xsl:value-of select="$protocol"/>
45                          </message>
46                      </xnm:error>
47                  </xsl:when>
48                  <xsl:otherwise>
49                  <xsl:for-each select="$out/physical-interface/logical-interface">
50                          <xsl:call-template name="intf"/>
51                      </xsl:for-each>
52                  </xsl:otherwise>
53              </xsl:choose>
54          </interface-information>
55      </op-script-results>
56  </xsl:template>
57  <xsl:template name="intf">
58      <xsl:variable name="status">
59          <xsl:choose>
60              <xsl:when test="admin-status='up' and oper-status='up'">
61                  <xsl:text> </xsl:text>
62              </xsl:when>
63              <xsl:when test="admin-status='down'">
64                  <xsl:text>offline</xsl:text>
65              </xsl:when>
66              <xsl:when test="oper-status='down' and ../admin-status='down'">
67                  <xsl:text>p-offline</xsl:text>
68              </xsl:when>
69              <xsl:when test="oper-status='down' and ../oper-status='down'">
70                  <xsl:text>p-down</xsl:text>
71              </xsl:when>
72              <xsl:when test="oper-status='down'">
73                  <xsl:text>down</xsl:text>
74              </xsl:when>
75              <xsl:otherwise>
76                  <xsl:value-of select="concat(oper-status, '/', admin-status)"/>
77              </xsl:otherwise>
78          </xsl:choose>
79      </xsl:variable>

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>
90          <logical-interface>
91            <name><xsl:value-of select="name"/></name>
92            <xsl:if test="string-length($desc)">
93              <admin-status><xsl:value-of select="$desc"/></admin-status>
94            </xsl:if>
95            <admin-status><xsl:value-of select="$status"/></admin-status>
96            <xsl:choose>
97              <xsl:when test="$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>
104         </logical-interface>
105     </xsl:template>
106   </xsl:stylesheet>
```

Line-by-Line Explanation of the Script

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 "Boilerplate for Op Scripts" on page 285.

```
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 emitted.

```
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 79 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

```
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;
        }
    }
}
```

Testing ex-interface.xsl

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

1. From "Customizing Output of the show interfaces terse Command" on page 320, copy the XSLT or SLAX script into a text file, and name the file ex-interface.xsl. Copy the ex-interface.xsl file to the /var/db/scripts/op directory on your routing platform.
2. Include the file ex-interface.xsl statement at the [edit system scripts op] hierarchy level. If you are using the SLAX version of the script, change the filename to *filename*.slax.

   ```
   [edit system scripts op]
   file ex-interface.xsl;
   ```

3. Issue the commit and-quit command.
4. Compare the output of the show interfaces terse and op ex-interface commands:

```
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 ex-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-->

user@host> op ex-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 ex-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
```

Finding LSPs to Multiple Destinations

This example checks for label-switched paths (LSPs) to multiple destinations.

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: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>
</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";
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: ' _  .;
               }
            }
         }
```

```
        }
    }
}
```

Testing ex-lsp.xsl

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

1. From "Finding LSPs to Multiple Destinations" on page 330, copy the XSLT or SLAX script into a text file, and name the file ex-lsp.xsl. Copy the ex-lsp.xsl file to the /var/db/scripts/op directory on your routing platform.
2. Include the file ex-lsp.xsl statement at the [edit system scripts op] hierarchy level. If you are using the SLAX version of the script, change the filename to *filename*.slax.

   ```
   [edit system scripts op]
   file ex-lsp.xsl;
   ```

3. Issue the commit and-quit command.
4. When you issue the op ex-lsp address *address* command, the output looks like this:

   ```
   user@R4> op ex-lsp address 10.168.215.0/24
   Found: 192.168.215.1 (10.168.215.1) –> R4>R1
   Found: 192.168.215.2 (10.168.215.2) –> R4>R2
   Name: R3
   Missing: 10.168.215.3
   Name: R5
   Missing: 10.168.215.4
   Name: R6
   Missing: 10.168.215.5
   ```

Importing and Exporting Files

Use the JUNOScript file-put and file-get operations to move a file to or from a remote server. Especially useful when the remote server sits outside a firewall, these commands move the files using the existing JUNOScript stream. This obviates the need to open a separate stream for the file transfer, handle authentication before moving files, and verify that the file transfer service is available.

Exporting Files to a Remote Server

Use the JUNOScript file-put command to transfer files within an existing remote JUNOScript connection. The basic syntax for using the file-put command is as follows:

```
<rpc>
    <file-put>
        <filename>value</filename>
        <encoding>value</encoding>
        <permission>value</permission>
        <delete-if-exist />
```

```
        <file-contents>file</file-contents>
    </file-put>
</rpc>
```

The following attributes are used with the file-put command. These attributes can be placed in any order with the exception of the file-contents attribute. The file-contents attribute must be the last attribute used with the file-put command.

- filename—(Mandatory) Within this tag, you include the full or relative file path and filename of the file for export. When you use a relative file path, the specified file path should be relative to the user's home directory. If the specified file directory does not exist, the system returns a "directory not found" error.
- encoding—(Mandatory) Specifies the type of encoding used. You can use ASCII or base64 encoding.
- 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.
- delete-if-exist—(Optional) If specified, an existing file on the remote server will be overwritten. If this attribute is not set, an error is returned if an existing file is encountered.
- file-contents—(Mandatory) The ASCII or base64 encoded file to be exported to the remote server. This attribute must be the last attribute used.

Importing Files from a Remote Server

The JUNOScript file-get command can be used to transfer files within an existing remote JUNOScript connection. Unless otherwise specified by the user, the imported file will be placed in the user's home directory. The basic syntax for using the file-get command is as follows:

```
<rpc>
    <file-get>
        <filename>value</filename>
        <encoding>value</encoding>
    </file-put>
</rpc>
```

The following attributes are used with the file-get command.

- filename—(Mandatory) Within this tags, you include the full or relative file path and filename of the file for import. When you use a relative file path, the specified file path is relative to the user's home directory.
- encoding—(Mandatory) Specifies the type of encoding used. You can use ASCII or base64 encoding.

Chapter 22

Summary of Op Script Configuration Statements

The following section explains each of the script configuration statements. The statements are organized alphabetically.

arguments

Syntax
```
arguments {
   argument-name;
}
```

Hierarchy Level [edit system scripts opfile *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS op scripts, configure command-line arguments to the script.

Usage Guidelines See “Declaring Arguments in Op Scripts” on page 305 .

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

command

Syntax command *filename-alias*;

Hierarchy Level [edit system scripts op file *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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.

Usage Guidelines See “Enabling an Operation Script and Defining a Script Alias” on page 304.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

description

Syntax description *descriptive-text*;

Hierarchy Level [edit system scripts op file *filename*],
[edit system scripts op file *filename* arguments *name*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS op scripts, provide a help-text string that appears in the command-line interface (CLI).

Usage Guidelines See "Declaring Arguments in Op Scripts" on page 305 and "Configuring Command-Line Help Text" on page 307.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh-from, source

file

Syntax

```
file filename {
   arguments {
      argument-name {
         description descriptive-text;
      }
   }
   command filename-alias;
   description descriptive-text;
   refresh;
   refresh-from url;
   source url;
}
```

Hierarchy Level [edit system scripts op]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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.

Usage Guidelines See "Configuring Operation Scripts" on page 303.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

op

Syntax

```
op {
   file filename {
      arguments {
         argument-name {
            description descriptive-text;
         }
      }
      command filename-alias;
      description descriptive-text;
      refresh;
      refresh-from url;
      source url;
   }
   refresh;
   refresh-from url;
   traceoptions {
      file filename <files number> <size size>;
      flag flag;
   }
}
```

Hierarchy Level [edit system scripts]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS op scripts, configure an operation scripting mechanism.

The statements are explained separately.

Usage Guidelines See “Configuring Operation Scripts” on page 303.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

refresh

Syntax refresh;

Hierarchy Level [edit system scripts op],
[edit system scripts op file *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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.

Usage Guidelines See "Refreshing an Op Script from the Master Source" on page 308.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh-from, source

refresh-from

Syntax refresh-from *url*;

Hierarchy Level [edit system scripts op],
[edit system scripts op file *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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.

Options *url*—The source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

Usage Guidelines See "Refreshing an Op Script from a Different Location" on page 309.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh, source

scripts

Syntax

```
scripts {
    op {
        file filename {
            arguments {
                argument-name {
                    description descriptive-text;
                }
            }
            command filename-alias;
            description descriptive-text;
            refresh;
            refresh-from url;
            source url;
        }
        refresh;
        refresh-from url;
        traceoptions {
            file filename <files number> <size size>;
            flag flag;
        }
    }
}
```

Hierarchy Level [edit system]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS op scripts, configure scripting mechanisms.

The statements are explained separately.

Usage Guidelines See “Configuring Operation Scripts” on page 303.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

source

Syntax source *url*;

Hierarchy Level [edit system scripts op file *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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 and commit the configuration, the local copy is overwritten by the version stored at the specified URL.

Options *url*—The source specified as an HTTP URL, FTP URL, or scp-style remote file specification.

Usage Guidelines See "Refreshing an Op Script from the Master Source" on page 308.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Related Topics refresh, refresh-from

traceoptions

Syntax

```
traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
}
```

Hierarchy Level [edit system scripts op]

Release Information Statement introduced in JUNOS Release 7.6.

Description Define tracing operations for op scripts.

Default If you do not include this statement, no op-script-specific tracing operations are performed.

Options *filename*—Name of the file to receive the output of the tracing operation. All files are placed in the directory /var/log. By default, op script process tracing output 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 op-script.log as the filename.

files *number*—(Optional) Maximum number of trace files. When a trace file named *trace-file* reaches its maximum size, it is renamed *trace-file*.0, then *trace-file*.1, and so on, until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum number of files, you also must specify a maximum file size with the size option and a filename.
Range: 2 through 1000
Default: 10 files

flag—Tracing operation to perform. To specify more than one tracing operation, include multiple flag statements. You can include the following flags:

- all—Log all operations
- events—Log important events
- input—Log op script input data
- offline—Generate data for offline development
- output—Log op script output data
- rpc—Log op script RPCs
- xslt—Log the XSLT library

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 *trace-file*.0. When *trace-file* again reaches its maximum size, *trace-file*.0

is renamed *trace-file*.1 and *trace-file* is renamed *trace-file*.0. 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: *x*k to specify KB, *x*m to specify MB, or *x*g to specify GB
Range: 10 KB through 1 GB
Default: 128 KB

no-world-readable—Restrict file access to owner. This is the default.

world-readable—Enable unrestricted file access.

Usage Guidelines See "Tracing Op Script Processing" on page 311.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Part 4
Event Policy

- Event Policy Overview on page 347
- Configuring Event Policy on page 351
- Event Policy Examples on page 377
- Summary of Event Policy Configuration Statements on page 389

Chapter 23

Event Policy Overview

This chapter provides the following sections:

- Introduction on page 347
- How Event Policies Work on page 348

Introduction

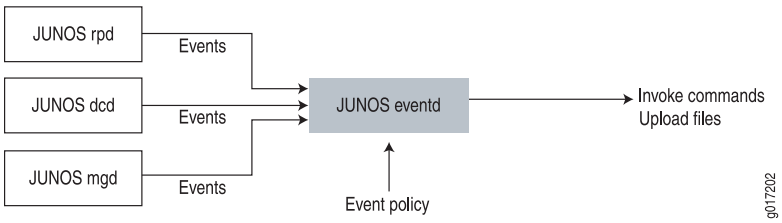
To diagnose a fault or error condition on a routing platform, you need relevant information about the state of the platform. You can derive state information from *event notifications*. Event notifications are system log messages and Simple Network Management Protocol (SNMP) traps. A JUNOS software process called the *event process* (eventd) receives event notifications—henceforth simply called *events*— from other JUNOS software processes.

Timely diagnosis and intervention can correct error conditions and keep the routing platform 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 routing platform 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 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 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 348 shows how the event process (eventd) interacts with other JUNOS software processes.

Figure 11: Eventd Process Interaction with Other JUNOS Software 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 ouput 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 *JUNOS CLI User Guide*.

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 software 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 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 JUNOScript 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 a Simple Network Management Protocol (SNMP) trap.

For more information, see “Configuring Event Policy” on page 351.

Chapter 24

Configuring Event Policy

Event policies can listen for specific events, create log files, invoke JUNOS 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] hierarchy level:

```
event-options {
    destinations {
        destination-name {
            transfer-delay seconds;
            archive-sites {
                url password password;
            }
        }
    }
    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 {
            event-script script-name {
                arguments {
                    parameter-name parameter-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-countnumber retry-interval seconds;
               transfer-delay seconds;
            }
            output-filename filename;
            output-format (text | xml);
            user-name username;
         }
         ignore;
         raise-trap;
         upload filename committed destination destination-name;
         upload filename filename destination destination-name {
            retry-countnumber 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;
   }
}
```

This chapter discusses the following topics:

- Defining Destinations for File Archiving on page 353
- Uploading Files on page 353
- Executing Operational Mode Commands on page 355
- Executing Event Scripts in an Event Policy on page 358
- Correlating Events on page 363
- Representing the Correlating Event in an Event Policy on page 365
- Triggering a Policy Based on Event Count on page 366
- Retrying the File Upload Action on page 366
- Configuring an Event to Be Ignored on page 367
- Using Regular Expressions to Refine the Set of Events That Cause a Policy to Be Executed on page 368
- Associating an Optional User with an Event Policy Action on page 369
- Assigning a Transfer Delay to an Event Policy Action on page 370
- Generating Internal Events on page 371
- Raising SNMP Traps on page 372

- Referencing Nonstandard Events on page 373
- Tracing Event Policy Processing on page 373

Defining Destinations for File Archiving

To upload information for analysis, first you define a destination for the information to be archived. To define destinations, include the following statements at the [edit event-options] hierarchy level:

```
[edit event-options]
destinations {
   destination-name {
      transfer-delay seconds;
      archive-sites {
         url password password;
      }
   }
}
```

After you define a destination, you can reference the destination in an event policy. For information about referencing destinations, see "Uploading Files" on page 353 and "Executing Operational Mode Commands" on page 355.

The optional 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. A transfer delay allows you to make sure a large file, such as a core file, is completely generated before the upload begins.

In the archive-sites statement, you can specify a destination as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification. URLs of the type file:// are not supported; however, local router directories are supported (for example, /var/tmp/). When you specify the archive site, do not add a forward slash (/) to the end of the URL. The format for the destination filename is *router-name_filename_YYYYMMDD_HHMMSS*.

Optionally, you can specify a plain-text password for login into an archive site.

For a configuration example, see "Correlating Events Based on Receipt of Other Events Within a Specified Time Interval" on page 377.

Uploading Files

Various types of files are useful in diagnosing an event. These files include system log files, core files, and configuration files. When a routing platform event occurs, you can upload relevant files to a specified location for analysis.

To configure a policy that uploads files, include the following statements at the [edit event-options] hierarchy level:

```
[edit event-options]
policy policy-name {
   events [ events ];
```

```
    then {
      upload filename committed destination destination-name;
      upload filename filename destination destination-name {
        retry-countnumber retry-interval seconds;
        transfer-delay seconds;
      }
    }
  }
```

When an event policy uploads files for analysis, the files are time-stamped to ensure the filenames are unique, as follows:

router-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:

router-name_filename_YYYYMMDD_HHMMSS_number

The number can be from 001 through 999. For example, if you have an event policy with output filename rpd-messages on router1, and this event policy is executed three times in 1 second, the files are named as follows:

- router1_rpd-messages_20070623_132333
- router1_rpd-messages_20070623_132333_001
- router1_rpd-messages_20070623_132333_002

In the events statement, you can list multiple events. If one or more of the listed events occurs, the upload action is executed. 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:

```
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 “Referencing Nonstandard Events” on page 373.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events” on page 371.

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, you can configure the following:

```
upload filename /var/log/messages*;
```

To upload the committed configuration file, include the committed statement in the configuration:

```
[edit event-options policy policy-name then]
upload filename committed destination destination-name;
```

In the destination statement, include the destination name that you configured at the [edit event-options destinations] hierarchy level. For more information, see "Defining Destinations for File Archiving" on page 353.

Executing Operational Mode Commands

Operational mode commands request that the router perform an operation or provide diagnostic output. They allow you to view statistics and information about a routing platform'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 Interfaces Command Reference*
- *JUNOS Routing Protocols and Policies Command Reference*
- *JUNOS System Basics and Services Command Reference*

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:

```
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 "Referencing Nonstandard Events" on page 373.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events" on page 371.

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 ouput 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 *JUNOS 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 events 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 router 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 "Defining Destinations for File Archiving" on page 353.

For a configuration example, see "Correlating Events Based on Receipt of Other Events Within a Specified Time Interval" on page 377.

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 JUNOScript remote procedure calls (RPCs). Additionally, you can pass to an event script a set of arguments that you define.

A script can change the router 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 run an event script by configuring 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 {
                parameter-name parameter-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:

```
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 "Referencing Nonstandard Events" on page 373.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events" on page 371.

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 when event scripts are located on the router's hard drive and in the /config/scripts/event directory when event scripts are located in the flash drive. For more information on event script file location, see "How Event Scripts Work" on page 416.Furthermore, the event scripts must be enabled at the [edit event-options event-script file] hierarchy level. For more information, see "Installing Event Scripts on a Router" on page 417 and "Event Policy Examples" on page 377.

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 *JUNOS 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 router 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 "Defining Destinations for File Archiving" on page 353.

For the output-filename and destination statements, there are four configuration scenarios:

Scenario 1 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.

Scenario 2 In the configuration, you can include the destination statement. 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.

Example: Scenario 2 On a router named router2, configure an event script action with a destination host, and omit the outut-filename statement. Define the destination host as ftp://user@router1//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@router1//tmp" password "$9$xkJNbYg4ZDH.oJ.fQnpuSyl"; ##
           SECRET-DATA***
      }
   }
}
```

In this example, The /var/cmd.txt file resides on router router2. The event policy uses the File Transfer Protocol (FTP) to upload this file to the /tmp directory on router router1.

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 router router1. The event policy renames the /var/cmd.txt file as router2_cmd.txt_*YYYYMMDD_HHMMSS_range*.

Scenario 3 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.

Scenario 4 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.

Correlating Events

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 not events [ events ];
   attributes-match {
      event1.attribute-name equals event2.attribute-name;
      event.attribute-name matches regular-expression;
      event1.attribute-name starts-with event2.attribute-name;
   }
}
```

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 "Referencing Nonstandard Events" on page 373.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events" on page 371.

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 Cause a Policy to Be Executed" on page 368.
- *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*.

You can include the attributes-match statement only if you include one or more within statements in the same policy configuration. This means the events are correlated only if they occur within a specified time period.

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 ouput 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 *JUNOS 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 (?).

For configuration examples, see "Ignoring Events Based on Receipt of Other Events" on page 385, "Correlating Events Based on Event Attributes" on page 385, and "Controlling Event Policy Using a Regular Expression" on page 385.

Representing the Correlating Event in an Event Policy

As described in "Executing Operational Mode Commands" on page 355, 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 "Correlating Events" on page 363, 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 "Representing the Correlating Event in an Event Policy" on page 380.

Triggering a 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.

This do this, include the optional trigger statement at the [edit event-options policy *policy-name* within *seconds*] hierarchy level:

```
[event-options policy policy-name within seconds]
trigger (on number | after number | until number);
```

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 *number*—The policy is executed when the number of matching events received equals *number* plus one.
- on *number*—The policy is executed when the number of matching events received equals *number*.
- until *number*—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals *number*.

For configuration examples, see "Triggering a Policy Based on Event Count" on page 383.

Retrying 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 *script-name* 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* destination *destination*]

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 configuration examples, see "Retrying the File Upload Action" on page 382.

Configuring an Event to Be Ignored

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:

```
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 "Referencing Nonstandard Events" on page 373.

In addition, you can reference internally generated events, which are discussed in "Generating Internal Events" on page 371.

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.

For configuration examples, see "Ignoring Events Based on Receipt of Other Events" on page 385 and "Dampening an Event" on page 387.

Using Regular Expressions to Refine the Set of Events That Cause a Policy to Be Executed

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 18 on page 368 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 18: 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.

Table 18: Regular Expression Operators for the matches Statement *(continued)*

Operator	Matches
? (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 the JUNOS software.
^ (caret)	The start of a line, when the caret appears outside square brackets. 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 "Controlling Event Policy Using a Regular Expression" on page 385.

Associating an Optional User with 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 router 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* [edit event-options policy then event-script *script-name*]
- [edit event-options policy *policy-name* then execute-commands]
- [edit event-options policy *policy-name* then upload filename *filename* destination *destination*]

NOTE: The username that you specify must be configured at the [edit system login] hierarchy level. For more information, see the *JUNOS System Basics Configuration Guide*.

For a configuration example, see "Associating an Optional User with an Event Policy Action" on page 381.

Assigning a Transfer Delay to an Event Policy Action

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 "Defining Destinations for File Archiving" on page 353, 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 {
        "http://robot@my.big.com/foo/moo" password "password";
        "http://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 *script-name* 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* destination *destination-name*]

If you configure a transfer delay in the destination definition (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 configuration examples, see "Assigning a Transfer Delay to an Event Policy Action" on page 378.

Generating Internal Events

Internal events are events you create yourself to trigger a policy to be executed. They are not generated by JUNOS software 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 be from 60 through 604,800 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.

For configuration examples, see "Generating an Internal Event Every Hour" on page 386 and "Generating an Internal Event at Midnight" on page 387.

Raising SNMP Traps

Simple Network Management Protocol (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. 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 valuable if you use NMS traps rather than system log messages to monitor your network.

To configure a 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]
event [ events ];
then {
    raise-trap;
}
```

A new MIB (jnx-syslog.mib) supports this policy action. For more information, see the *JUNOS Network Management Configuration Guide*.

For a configuration example, see “Raising an SNMP Trap in Response to an Event” on page 388.

Referencing Nonstandard Events

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 19 on page 373.

Table 19: Event ID by System Log Message Origin

Event IDs	Origin
SYSTEM	Messages from UNIX domain sockets
KERNEL	Messages from the kernel
PIC	Messages from PICs
PFE	Messages from the Packet Forwarding Engine
LCC	On a TX Matrix platform, messages from as line-card chassis (LCC)
SCC	On a TX Matrix platform, messages from a switch-card chassis (SCC)

To base your event policy on the event types shown in Table 19 on page 373, 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 “Referencing Nonstandard Events” on page 388.

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 eventd.0, then eventd.1, and so on, until there are three trace files. Then the oldest trace file

(eventd.2) is overwritten. (For more information about how log files are created, see the *JUNOS 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) in which trace files are located. 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> <size size> <world-readable | no-world-readable> <match
    regex>;
flag {
    all;
    configuration;
    database;
    events;
    policy;
    server;
    syslog
    timer-events;
}
```

These statements are described in the following sections:

- Configuring the Event Policy Log Filename on page 374
- Configuring the Number and Size of Event Policy Log Files on page 374
- Configuring Access to the Log File on page 375
- Configuring a Regular Expression for Lines to Be Logged on page 375
- Configuring the Trace Operations on page 375

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] hierarchy level:

```
[edit event-options traceoptions]
file 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 *filename*.0 and a new file called *filename* is created.

When the new *filename* reaches 2 MB, *filename*.0 is renamed *filename*.1 and *filename* is renamed *filename*.0. This process repeats until there are 20 trace files. Then the oldest file (*filename*.19) is overwritten.

The number of files can be from 2 through 1000 files. The file size of each file can be 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 file world-readable statement at the [edit event-options traceoptions] hierarchy level:

```
[edit event-options traceoptions]
file world-readable;
```

To explicitly set the default behavior, include the file no-world-readable statement at the [edit event-options traceoptions] hierarchy level:

```
[edit event-options traceoptions]
file 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 (*regex*) to be matched:

```
[edit event-options traceoptions]
file filename match regex;
```

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;
    configuration;
    events;
    database;
    policy;
    server;
```

```
    timer-events;
}
```

Table 20 on page 376 describes the meaning of the event policy tracing flags.

Table 20: 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
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
```

Chapter 25

Event Policy Examples

This chapter includes the following examples:

- Correlating Events Based on Receipt of Other Events Within a Specified Time Interval on page 377
- Assigning a Transfer Delay to an Event Policy Action on page 378
- Representing the Correlating Event in an Event Policy on page 380
- Associating an Optional User with an Event Policy Action on page 381
- Retrying the File Upload Action on page 382
- Triggering a Policy Based on Event Count on page 383
- Ignoring Events Based on Receipt of Other Events on page 385
- Correlating Events Based on Event Attributes on page 385
- Controlling Event Policy Using a Regular Expression on page 385
- Generating an Internal Event Every Hour on page 386
- Generating an Internal Event at Midnight on page 387
- Dampening an Event on page 387
- Raising an SNMP Trap in Response to an Event on page 388
- Referencing Nonstandard Events on page 388

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;
}
```

You configure this policy as follows. In this example, two archive sites are specified. The router attempts to transfer to the first archive site in the list, moving to the next site only if the transfer fails.

```
event-options {
    policy 1 {
        events [ event3 event4 event5 ];
        within 60 events [ event1 event2 ];
        then {
            execute-commands {
                commands {
                    "command";
  "command";
  "command";
                }
                output-filename my_cmd_out;
                destination policy-1-command-dest;
            }
        }
        destinations {
            policy-1-command-dest {
                archive-sites {
                    http://robot@my.big.com/a/b;
                    http://robot@my.little.com/a/b;
                }
            }
        }
    }
}
```

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 daemon.core file to the some-dest destination. The policy2 event policy has no transfer delay when uploading the daemon.core file to the same destination.

```
event-options {
    policy policy1 {
        events e1;
        then {
            upload filename daemon.core destination some-dest {
                transfer-delay 5;
            }
        }
    }
    policy policy2 {
        events e2;
        then {
            upload filename daemon.core destination some-dest;
        }
    }
    destinations {
```

```
        some-dest {
            archive-sites {
                "http://robot@my.little.com/foo/moo" password "password";
                "http://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 daemon.core file to the destination. The policy2 event policy has a 2-second transfer delay when uploading the daemon.core file to the destination.

```
event-options {
    policy policy1 {
        events e1;
        then {
            upload filename daemon.core destination some-dest {
                transfer-delay 5;
            }
        }
    }
    policy policy2 {
        events e2;
        then {
            upload filename daemon.core destination some-dest;
        }
    }
    destinations {
        some-dest {
            transfer-delay 2;
            archive-sites {
                "http://robot@my.little.com/foo/moo" password "password";
                "http://robot@my.big.com/foo/moo" password "password";
            }
        }
    }
}
```

Example 3 The policy1 event-policy is executed with user1 privileges and uploads the daemon.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 daemon.core file after a transfer delay of 6 seconds (4 seconds + 2 seconds).

```
event-options {
    policy policy1 {
        events e1;
        then {
            upload filename daemon.core destination some-dest {
                transfer-delay 5;
                user-name user1;
            }
        }
    }
    policy policy2 {
```

```
        events e2;
        then {
            upload filename daemon.core destination some-dest {
                transfer-delay 4;
            }
        }
    }
    destinations {
        some-dest {
            transfer-delay 2;
            archive-sites {
                "http://robot@my.little.com/foo/moo" password "password";
                "http://robot@my.big.com/foo/moo" password "password";
            }
        }
    }
}
```

Representing the Correlating Event in an Event Policy

Consider the following example:

```
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 {$$.interface-name} command, {$$.interface-name} is substituted with the value of the interface-name attribute of event e1, e2, or e3.

In the show interfaces {$e4.interface-name} command, {$e4.interface-name} is substituted with the value of the interface-name attribute of the most recent e4 event.

In the show interfaces {$*.interface-name} command, {$*.interface-name} is substituted with the value of the interface-name attribute of the most recent e4, e5, or e6 event. If e4 is received within 60 seconds of e1, e2, or e3, {$*.interface-name} is substituted with the value of the interface-name attribute for e4. Otherwise, if e5 is received within 60 seconds of e1, e2, or e3, {$*.interface-name} is substituted with the value of the interface-name attribute for e5. Else, if e6 is received within 60 seconds of e1, e2, or e3, {$*.interface-name} is substituted with the value of the interface-name

attribute for e6. If the correlating event (e4, e5, or e6) does not have an interface-name attribute, the software does not execute the show interfaces {$*.interface-name} command.

If both e4 and e5 are received within 60 seconds of e1, then {$*.interface-name} is substituted with the value of the interface-name attribute for e4. 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.

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.

```
system {
    login {
        user user1 {
            class operator;
        }
    }
}
event-options {
    policy policy1 {
        events e1;
        then {
            execute-commands {
                commands {
                    "show version";
                }
                user-name user1;
                output-filename command-output.txt;
                destination some-dest;
            }
        }
    }
    policy policy2 {
        events e2;
        then {
            event-script script.xsl {
                output-filename event-script-output.txt;
                destination some-dest;
            }
        }
    }
}
```

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.

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 valid JUNOS software 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.

```
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 21 on page 384 shows how events add to the count.

Table 21: 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 21 on page 384 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.

```
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;
            SSH_LOGIN_FAIL.username matches roger;
        }
        then {
            event-script login.xsl {
                destination some-dest;
            }
        }
    }
}
```

Ignoring Events Based on Receipt of Other Events

In the following policy, if *event1*, *event2*, or *event3* has been received, and either *event4* or *event5* has been received within the last 600 seconds, and *event6* has not been received 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.

```
event-options {
    policy 1 {
        events [ event1 event2 event3 ];
        within 600 events [ event4 event5 ];
        within 800 not events event6;
        then {
            ignore;
        }
    }
}
```

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.

```
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;
        }
    }
}
```

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.

```
event-policy {
    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;
            }
        }
    }
}
```

Generating an Internal Event Every Hour

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.

```
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 {
            ...
        }
    }
}
```

Generating an Internal Event at Midnight

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.

```
event-options {
    generate-event it-is-midnight time-of-day 00:00:00;
    policy midnight-chores {
        events it-is-midnight;
        then {
            ...
        }
    }
}
```

Dampening an Event

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.

```
event-options {
    policy dampen-policy {
        events event1;
        within 60 events event1;
        then {
            ignore;
        }
    }
    policy policy {
        events event1;
        then {
            ...
        }
    }
}
```

Raising an SNMP Trap in Response to an Event

Raise a trap and execute an associated event script in response to an event:

```
event-options {
    policy p1 {
        events ui_mgd_terminate;
        then {
            raise-trap;
            event-script bgp.xsl {
                arguments {
                    destination {$ui_mgd_terminate.destination};
                    code 2;
                }
                output-filename bgp-out;
                destination bsd3;
            }
        }
    }
}
```

Referencing Nonstandard Events

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.

```
event-options {
    policy kernel-policy {
        events KERNEL;
        attributes-match {
            KERNEL.message matches "exited on signal 11";
        }
        then {
            raise-trap;
        }
    }
}
```

Chapter 26

Summary of Event Policy Configuration Statements

The following sections explain each of the event policy configuration statements. 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 Release 7.5.

Description Specify an archive site to which files are transferred. If you specify more than one archive site, the router 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 Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification. URLs of the type file:// are not supported; however, local router directories are supported (for example, /var/tmp/).

password *password*—A plain-text password for login into the archive site.

Usage Guidelines See "Defining Destinations for File Archiving" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

arguments

Syntax
```
arguments {
    argument-name;
}
```

Hierarchy Level [edit event-options policy *policy-name* then event-script *filename*]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS op scripts, include command-line arguments to the script.

Usage Guidelines See "Executing Event Scripts in an Event Policy" on page 358.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

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 Release 7.5.

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.

The statements are explained separately.

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

commands

Syntax

```
commands {
   "command";
}
```

Hierarchy Level [edit event-options policy *policy-name* then execute-commands]

Release Information Statement introduced in JUNOS Release 7.5.

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.

Usage Guidelines See “Executing Operational Mode Commands” on page 355 and “Representing the Correlating Event in an Event Policy” on page 365.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

destination

See the following sections:

- destination (Command or Script Output) on page 392
- destination (Routing Platform Files) on page 392

destination (Command or Script Output)

Syntax destination *destination-name*;

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 Release 7.5.
[edit event-options policy *policy-name* then event-script *filename*] hierarchy level added in JUNOS Release 7.6.

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.

Usage Guidelines See "Executing Operational Mode Commands" on page 355 and "Executing Event Scripts in an Event Policy" on page 358.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

destination (Routing Platform Files)

Syntax destination *destination-name*;

Hierarchy Level [edit event-options policy *policy-name* then upload filename committed],
[edit event-options policy *policy-name* then upload filename *filename*]

Release Information Statement introduced in JUNOS Release 7.5.

Description Assign a location to which to upload routing platform files, such as configuration files, core files, and system log files.

Options *destination-name*—Name of a destination defined in the destinations statement.

Usage Guidelines See "Uploading Files" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

destinations

Syntax

```
destinations {
   destination-name {
      transfer-delay seconds;
      archive-sites {
         url password password;
      }
   }
}
```

Hierarchy Level [edit event-options]

Release Information Statement introduced in JUNOS Release 7.5.

Description Define a destination 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 statements are explained separately.

Usage Guidelines See "Defining Destinations for File Archiving" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

equals

Syntax *event1.attribute-name* equals *event2.attribute-name*;

Hierarchy Level [edit event-options policy *policy-name* attributes-match]

Release Information Statement introduced in JUNOS Release 7.5.

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.

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

event-options

Syntax

```
event-options {
    destinations {
        destination-name {
            transfer-delay seconds;
            archive-sites {
                url password password;
            }
        }
    }
    generate-event event-name {
        time-interval seconds;
        time-of-day hh:mm:ss;
    }
    policy policy-name {
        events [ events ];
        within seconds not events [ events ];
        attributes-match {
            event1.attribute-name equals event2.attribute-name;
            event.attribute-name matches regular-expression;
            event1.attribute-name starts-with event2.attribute-name;
        }
        then {
            event-script filename {
                arguments {
                    name;
                }
                output-filename filename;
                destination destination-name;
            }
            execute-commands {
                commands {
                    "command"
                }
                output-filename filename;
                output-format (text | xml);
                destination destination-name;
            }
            ignore;
            raise-trap;
            upload filename committed destination destination-name;
            upload filename filename destination destination-name;
        }
    }
    traceoptions {
        file filename <files number> <size size> world-readable | no-world-readable>;
        flag flag;
    }
}
```

Hierarchy Level [edit]

Release Information Statement introduced in JUNOS Release 7.5.

Description Configure event policies.

The statements are explained separately.

Usage Guidelines See "Configuring Event Policy" on page 351.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

event-script

Syntax
```
event-script filename {
    arguments {
        name;
    }
    output-filename filename;
    destination destination-name;
}
```

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 7.6.

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.

Usage Guidelines See "Executing Event Scripts in an Event Policy" on page 358.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

events

See the following sections:

- events (Associating Events with a Policy) on page 396
- events (Correlating Events with Each Other) on page 396

events (Associating Events with a Policy)

Syntax events [*events*];

Hierarchy Level [edit event-options policy *policy-name*]

Release Information Statement introduced in JUNOS Release 7.5.

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 software processes.

Usage Guidelines See "Configuring Event Policy" on page 351.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

events (Correlating Events with Each Other)

Syntax within *seconds* not events [*events*];

Hierarchy Level [edit event-options policy *policy-name*]

Release Information Statement introduced in JUNOS 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.

Options [*events*]—List of events. Events can be internally generated or they can be generated by JUNOS software processes.

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

execute-commands

Syntax
```
execute-commands {
    commands {
        "command";
    }
    output-filename filename;
    output-format (text | xml);
    destination destination-name;
}
```

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 7.5.

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.

Usage Guidelines See "Executing Operational Mode Commands" on page 355.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

filename

Syntax
```
filename filename;
    filename committed;
```

Hierarchy Level [edit event-options policy *policy-name* then upload]

Release Information Statement introduced in JUNOS Release 7.5.

Description Specify a routing platform file to be uploaded.

Options committed—Upload the committed configuration file.

filename—Name of a file to be uploaded. You can specify multiple files with one filename configuration statement.

Usage Guidelines See "Uploading Files" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

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 Release 7.5.

Description Generate an internal event, based on a time interval or the time of day.

Options *event-name*—Name of an internally generated event.

The statements are explained separately.

Usage Guidelines See "Generating Internal Events" on page 371.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

ignore

Syntax ignore;

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 7.5.

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.

Usage Guidelines See "Configuring an Event to Be Ignored" on page 367.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

matches

Syntax *event.attribute-name* matches *regular-expression*;

Hierarchy Level [edit event-options policy *policy-name* attributes-match]

Release Information Statement introduced in JUNOS 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.

Usage Guidelines See "Correlating Events" on page 363 and "Using Regular Expressions to Refine the Set of Events That Cause a Policy to Be Executed" on page 368.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

not

Syntax within *seconds* not events [*events*];

Hierarchy Level [edit event-options policy *policy-name*]

Release Information Statement introduced in JUNOS Release 7.5.

Description Specify a policy that takes effect only if events do not occur within a time interval.

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

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 Release 7.5.
Support at the [edit event-options policy *policy-name* then event-script *script-name*] hierarchy level introduced in JUNOS 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.

Usage Guidelines See "Executing Operational Mode Commands" on page 355 and "Executing Event Scripts in an Event Policy" on page 358.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

output-format

Syntax output-format (text | xml);

Hierarchy Level [edit event-options policy *policy-name* then event-script *script-name*],
[edit event-options policy *policy-name* then execute-commands]

Release Information Statement introduced in JUNOS Release 7.5.
Support at the [edit event-options policy *policy-name* then event-script *script-name*] hierarchy level introduced in JUNOS Release 8.3.

Description Assign a format for the output of the specified commands. By default, the 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 *script-name*] hierarchy level.

Options text—Formatted ASCII text.

xml—JUNOS Extensible Markup Language (XML) tags.

Usage Guidelines See "Executing Operational Mode Commands" on page 355 and "Executing Event Scripts in an Event Policy" on page 358.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

policy

Syntax

```
policy policy-name {
   events [ events ];
   within seconds not events [ events ];
   attributes-match {
      event1.attribute-name equals event2.attribute-name;
      event.attribute-name matches regular-expression;
      event1.attribute-name starts-with event2.attribute-name;
   }
   then {
      event-script filename {
         arguments {
            name;
         }
         output-filename filename;
         destination destination-name;
      }
      execute-commands {
         commands {
            "command";
         }
         output-filename filename;
         output-format (text | xml);
         destination destination-name;
      }
      ignore;
      raise-trap;
      upload filename committed destination destination-name;
      upload filename filename destination destination-name;
   }
}
```

Hierarchy Level [edit event-options]

Release Information Statement introduced in JUNOS Release 7.5.

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.

Usage Guidelines See “Configuring Event Policy” on page 351.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

raise-trap

Syntax raise-trap;

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 8.1.

Description Define a policy that raises a Simple Network Management Protocol (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.

Usage Guidelines See "Raising SNMP Traps" on page 372.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

retry-count

Syntax retry-count *number* retry-interval *seconds*;

Hierarchy Level [edit event-options policy *policy-name* then event-script *script-name* 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* destination *destination*]

Release Information Statement introduced in JUNOS Release 8.4.

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.

Usage Guidelines See "Retrying the File Upload Action" on page 366.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

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 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.

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

then

Syntax

```
then {
    event-script filename {
        arguments {
            name;
        }
        output-filename filename;
        destination destination-name;
    }
    execute-commands {
        commands {
            "command";
        }
        output-filename filename;
        output-format (text | xml);
        destination destination-name;
    }
    ignore;
    raise-trap;
    upload filename committed destination destination-name;
    upload filename filename destination destination-name;
}
```

Hierarchy Level [edit event-options policy *policy-name*]

Release Information Statement introduced in JUNOS Release 7.5.

Description Define actions to take if an event occurs. For each policy, you can configure multiple actions.

The statements are explained separately.

Usage Guidelines See “Configuring Event Policy” on page 351.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

time-interval

Syntax time-interval *seconds*;

Hierarchy Level [edit event-options generate-event *event-name*]

Release Information Statement introduced in JUNOS Release 7.5.

Description Configure a frequency at which to generate a particular event.

Options *seconds*—Time interval between internally generated events.
Range: 60 through 604,800 seconds

Usage Guidelines See "Generating Internal Events" on page 371.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

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 Release 7.5.

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.

Usage Guidelines See "Generating Internal Events" on page 371.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

traceoptions

Syntax

```
traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable> <match
        regex>;
    flag flag;
}
```

Hierarchy Level [edit event-options]

Release Information Statement introduced in JUNOS Release 7.5.

Description Define tracing operations for event policy.

Default If you do not include this statement, no event-policy-specific tracing operations are performed.

Options *filename*—Name of the file to receive the output of the tracing operation. All files are placed in the directory /var/log. By default, commit script process tracing output is placed in the file eventd. If you include the file statement, you must specify a filename. To retain the default, you can specify eventd as the filename.

files *number*—(Optional) Maximum number of trace files. When a trace file named *trace-file* reaches its maximum size, it is renamed *trace-file*.0, then *trace-file*.1, and so on, 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—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
- policy—Log policy processing
- server—Log communication with processes that are generating events
- timer-events—Log internally generated events

match *regex*—(Optional) Refine the output to include lines that contain the regular expression.

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 *trace-file*.0. When the *trace-file* again reaches its maximum size, *trace-file*.0 is renamed *trace-file*.1 and *trace-file* is renamed *trace-file*.0. 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: *x*k to specify KB, *x*m to specify MB, or *x*g to specify GB
Range: 10 KB through 1 GB
Default: 128 KB

world-readable—(Optional) Enable unrestricted file access.

Usage Guidelines See "Tracing Event Policy Processing" on page 373.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

transfer-delay

Syntax transfer-delay *seconds*;

Hierarchy Level [edit event-options destinations *destination-name*],
[edit event-options policy *policy-name* then event-script *script-name* 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* destination *destination-name*]

Release Information Statement introduced in JUNOS Release 7.5.
Support at the [edit event-options policy *policy-name* then ...] hierarchy levels introduced in JUNOS Release 8.4.

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 uploading files.

Usage Guidelines See "Defining Destinations for File Archiving" on page 353 and "Assigning a Transfer Delay to an Event Policy Action" on page 370.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

trigger

Syntax trigger (on *number* | after *number* | until *number*);

Hierarchy Level [event-options policy *policy-name* within *seconds*]

Release Information Statement introduced in JUNOS Release 8.4.

Description Configure an event policy to be triggered if an event or set of events occurs *N* 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 *number*—The policy is executed when the number of matching events received equals *number* + 1.

on *number*—The policy is executed when the number of matching events received equals *number*.

until *number*—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals *number*.

Usage Guidelines See "Triggering a Policy Based on Event Count" on page 366.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

upload

See the following sections:

- upload (Committed Configuration File) on page 410
- upload (Specified File) on page 410

upload (Committed Configuration File)

Syntax upload filename committed destination *destination-name*;

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 8.1.

Description On receipt of an event, upload the committed configuration file to a destination.

The statements are explained separately.

Usage Guidelines See "Uploading Files" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

upload (Specified File)

Syntax upload filename *filename* destination *destination-name*;

Hierarchy Level [edit event-options policy *policy-name* then]

Release Information Statement introduced in JUNOS Release 7.5.

Description On receipt of an event, specify a file to be uploaded to a destination.

The statements are explained separately.

Usage Guidelines See "Uploading Files" on page 353.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

user-name

Syntax user-name *username*;

Hierarchy Level [edit event-options policy *policy-name* then event-script *script-name*],
[edit event-options policy *policy-name* then execute-commands],
[edit event-options policy *policy-name* then upload filename *filename* destination *destination*]

Release Information Statement introduced in JUNOS Release 8.4.

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*—A username that is configured at the [edit system login] hierarchy level.

Usage Guidelines See "Associating an Optional User with an Event Policy Action" on page 369.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

within

Syntax within *seconds* not events [*events*];

Hierarchy Level [edit event-options policy *policy-name*]

Release Information Statement introduced in JUNOS 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

Usage Guidelines See "Correlating Events" on page 363.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Part 5
Event Scripts

- Event Scripts Overview on page 415
- Introduction to Writing Event Scripts on page 419
- Configuring Event Scripts on page 439
- Event Script Examples on page 447
- Summary of Event Script Configuration Statements on page 449

Chapter 27

Event Scripts Overview

JUNOS event scripts automate network and router management and troubleshooting. Event scripts can perform functions available through the remote procedure calls (RPCs) supported by either of the two application programming interfaces (APIs): the JUNOS Extensible Markup Language (XML) API and the JUNOScript API.

Event scripts allow you to do the following things:

- Diagnose and fix network problems automatically.
- Monitor the overall status of a routing platform.
- Reconfigure the routing platform to avoid or work around known problems in the JUNOS software.
- Change the router's configuration in response to a problem.

This chapter includes the following topics:

- Event Script Programming on page 415
- Event Scripts on page 416
- Storing Event Scripts on page 416
- How Event Scripts Work on page 416
- Using Event Scripts on page 417

Event Script Programming

Event scripts are based on two APIs: the JUNOS XML API and the JUNOScript API. For more information on the JUNOS XML API and the JUNOScript API, see "Introduction to the JUNOS XML and JUNOScript APIs" on page 9. 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 "Understanding XSLT" on page 15. For more information about SLAX, see "Understanding SLAX" on page 51.

Event Scripts

Event scripts—Automatically execute operational commands and inspect router output when triggered by previously configured event policy scripts. Event scripts are executed by the eventd process.

Storing Event Scripts

By default, commit, operation, and event scripts are stored on the router's hard drive. However, you can save these scripts to the flash memory by including the load-scripts-from-flash statement at the [edit system scripts] hierarchy level:

```
[edit system scripts]
load-scripts-from-flash;
```

The load-scripts-from-flash statement applies to all commit, operation, and event scripts, and changes the physical location of these scripts. This statement does not affect script operation. Event scripts located on the router's hard drive are accessed from the /var/db/scripts/event directory. Event scripts located on the router's CompactFlash drive are accessed from the /config/scripts/event directory.

You can view the currently active event scripts on the router in the /var/run/scripts/event directory. This directory allows you to view the active scripts without having to search for this information within the current configuration.

NOTE: When you switch from storing the files on the hard drive to storing files on the flash drive, or you switch from storing the files on the flash drive to storing files on the hard drive, you must manually move any scripts residing in the former memory location to the new memory location. If you do not do this, the scripts are not available to the router.

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 Policy Overview" on page 347 and "Executing Event Scripts in an Event Policy" on page 358.

You can use event scripts to generate changes to the router 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 routing platform configuration.

Using Event Scripts

This section provides directions for using event scripts on your router.

- Installing Event Scripts on a Router on page 417
- Replacing Event Scripts on page 417

Installing Event Scripts on a Router

To install event scripts on the router, follow these steps:

1. Write an event script.

   For more information on writing event scripts, see "Introduction to Writing Event Scripts" on page 419. For event script examples, see "Event Script Examples" on page 447.

2. Copy the script to the /var/db/scripts/event directory for the hard drive or the /config/script/event directory for the flash drive. Only users in the superuser JUNOS login class can access and edit files in the /var/db/scripts/event and /config/script/event directories.

NOTE: If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/event or /config/script/event directories on each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

3. Add the event script to the [edit event-options event-script] hierarchy level by including the file statement. Only users in the superuser class can add event scripts to the configuration.

   ```
   [edit event-options event-script]
   file filename;
   ```

4. Issue the commit command.

   The event script is now loaded into memory and ready for execution. The event script is executed automatically in response to system log events. For more information, see "Executing Event Scripts in an Event Policy" on page 358

The JUNOS software provides several tools to manage and monitor event script operations, including tracing operations. For more information, see "Configuring Event Scripts" on page 439.

Replacing Event Scripts

You do not need to change the router's configuration to update or replace an existing event script. You can replace event scripts in the /var/db/scripts/event or the

/config/scripts/event directories and then refresh the scripts within the eventd process using the the request system scripts event-scripts reload command. This command allows you to update event scripts without disrupting the operation of the router.

To replace event scripts, follow these steps:

1. Edit or write the new event script.

   For more information on writing event scripts, see "Introduction to Writing Event Scripts" on page 419.

2. Copy the new script to the /var/db/scripts/event directory on the hard drive or the /config/scripts/event directory on the flash drive. Only users in the superuser class can add event scripts to the configuration.

3. In the CLI operational mode, issue the request system scripts event-scripts reload command.

   Once issued, the request system scripts event-scripts reload command will reload all event scripts back into the eventd process.

Chapter 28

Introduction to Writing Event Scripts

When you write event scripts, you can use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided by the JUNOS software. These tools include basic boilerplate that you must include in all event scripts and an import file called junos.xsl, which includes several named templates and extension functions that make event scripts easier to read and write.

Event scripts are based on JUNOScript and JUNOS XML tag elements. Like all XML elements, angle brackets enclose the name of a JUNOScript or JUNOS XML 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 Juniper Networks documentation to indicate optional parts of CLI command strings.

This chapter includes the following topics:

- Boilerplate for Event Scripts on page 419
- Displaying Operational Mode Fields in XML on page 421
- Using RPCs and Operational Mode Commands on page 422
- Capturing and Using Event Details on page 423
- Importing the junos.xsl File on page 425

Boilerplate for Event Scripts

This section contains basic XSLT and SLAX script boilerplate for event scripts. You must include either XSLT or SLAX boilerplate as the starting point for all event scripts that you create. The XSLT boilerplate is followed by line-by-line comments.

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="/">
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), 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 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 them all prevents errors if the namespace 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 ../import/junos.xsl, which ships as part of the JUNOS software (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 “Importing the junos.xsl File” on page 425.

```
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 JUNOScript Remote Procedure Calls (RPCs). For more information, see “XPath” on page 17.

```
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>
```

After Line 11, you can define additional XSLT templates that are called from within the <xsl:template match="/"> template.

```
<!- - ... insert additional template definitions here ... - ->
```

Line 12 closes the style sheet and the event script.

```
12   </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 / {
   <event-script-results> {
      /*
         * Insert your code here.
      */
   }
}
```

Displaying Operational Mode Fields in XML

To write event scripts, you must gather information about the JUNOS XML version of operational mode commands and output fields. You can do this by consulting the *JUNOS XML API Operational Reference*.

Another very useful tool is 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/7.6R1/junos">
    <interface-information
xmlns="http://xml.juniper.net/junos/7.6I0/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

Most operational mode commands supported in the JUNOS software have XML equivalents known as *remote procedure calls* (RPCs). All operational mode commands that have XML equivalents are listed in the *JUNOS XML API Operational Reference*.

Using RPCs

RPCs can be invoked from event scripts. For writing cross-box functionality event scripts, a user can configure the username and passphrase for a remote machine in the [edit event-options event-script file *filename* remote-execution] hierarchy level.

```
[edit event-options event-script file filename remote-execution]
server-name {
   username username
   passphrase passphrase
}
```

These details for the configured remote machine, username, and passphrase are passed as input to the event script along with the event details. The event script can then utilize the jcs:open() function with these details as input parameters to open a connection to the remote machine on which to execute the RPCs. For details on this function see "jcs:open() Function" on page 429. Once a connection handle has been obtained, the event script can execute an RPC. To use an RPC in the event script, include the RPC in a variable declaration, as shown in the following snippet. This snippet is expanded and fully described in "Customizing Output of the show interfaces terse Command" on page 320.

```
<xsl:variable name="rpc">
   <get-interface-information/> # JUNOS RPC for the show interfaces command
</xsl:variable>
<xsl:variable name="out" select="jcs:invoke($rpc)"/>
...
```

Using Operational Mode Commands

Some operational mode commands do not currently have XML equivalents (RPCs). 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 with the | display xml piped command:

```
user@host> operational-mode-command | display xml
```

If the output of this command contains no more detail than the <output> and <cli> tag elements, the command might not have an XML equivalent. For example, the show host command has no XML equivalent. The only child elements of the <rpc-reply> element are the <output> and <cli> tag elements:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/7.6R1/junos">
  <output>
     ...
   </output>
   <cli>
      <banner></banner>
```

```
    </cli>
</rpc-reply>
```

NOTE: For some commands, if there is no corresponding configuration, the output of the piped | display xml command might contain no information, even though an RPC exists for the command. For example, suppose a router configuration contains no statements at the [edit class-of-service] hierarchy level. If you issue the show services cos statistics forwarding-class | display xml command on this router, the output does not contain the existing <service-cos-forwarding-class-statistics> response tag:

```
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, checking the *JUNOS XML API Operational Reference* is the most reliable method for determining whether a command has an XML equivalent.

You can reference commands that have no XML equivalent. You can do this by including the <command>, <xsl:value-of>, and <output> elements, as shown in the following snippet. This code snippet is expanded and fully described in "Displaying DNS Hostname Information" on page 317.

```
<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

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.

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. In addition, remote execution details will also be included, allowing the event script to execute remote procedure calls as detailed in "Using RPCs and Operational Mode Commands" on page 422. Event details are forwarded to the event script as XML in the following format:

```
<event-script-input>
    <trigger-event>
        <id>event id</id>
        <type>event type</type>
        <generation-time>time stamp</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>Name of the attribute</name>
                <value>Value of the attribute</value>
            </attribute>
        </attribute-list>
    </trigger-event>
    <received-events>
         <received-event>
            <id>event id</id>
            <type>event type</type>
                <generation-time>time stamp</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>Name of the attribute</name>
                        <value>Value of the attribute</value>
                    </attribute>
                 </attribute-list>
       <received-event>
    </received-events>
    <remote-execution-details>
        <remote-execution-detail>
            <remote-machine>machine name</remote-machine>
            <username>name</username>
           <passphrase>passphrase</passphrase>
        </remote-execution-detail>
    </remote-execution-details>
</event-script-input>
```

The section “Limiting Event Script Output Based on a Specific Event Type” on page 447 shows one method for using event details.

Importing the junos.xsl File

The import file junos.xsl contains several useful extension functions that you can call from an event script. To use these functions, you must include two instructions in your script: the jcs namespace mapping and the <xsl:import> element:

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

Both the namespace mapping and the <xsl:import> element are contained in the basic script boilerplate presented in "Boilerplate for Event Scripts" on page 419.

For more information, see the following sections:

- Extension Functions in the junos.xsl File on page 425
- Templates in the junos.xsl File on page 434

Extension Functions in the junos.xsl File

Extension functions in the junos.xsl file use the jcs prefix to avoid conflicting with standard XSLT functions. The functions in the jcs namespace allow you to accomplish scripting tasks more easily. To use these functions in your scripts, you simply include a *variable declaration*, a variable call with the select="jcs:*function*()" attribute, and pass in any required or optional arguments.

These functions are discussed in more detail in the following sections:

- jcs:break-lines() Function on page 426
- jcs:close() Function on page 426
- jcs:dampen() Function on page 426
- jcs:empty() Function on page 427
- jcs:execute() Function on page 427
- jcs:first-of() Function on page 427
- jcs:getsecret() Function on page 428
- jcs:hostname() Function on page 428
- jcs:input() Function on page 429
- jcs:invoke() Function on page 429
- jcs:open() Function on page 429
- jcs:output() Function on page 430
- jcs:parse-ip() Function on page 430
- jcs:printf() Function on page 431
- jcs:progress() Function on page 431
- jcs:regex() Function on page 431
- jcs:sleep() Function on page 432

- jcs:split() Function on page 432
- jcs:sysctl() Function on page 432
- jcs:syslog() Function on page 432
- jcs:trace() Function on page 434

jcs:break-lines() Function

The jcs:break-lines() function is used to break a single element into multiple elements, delimited by <newline> characters. This function is especially useful in dealing with large output elements, such as those returned using the show pfe command.

The function syntax is as follows:

```
var $lines jcs:break-lines($output);
```

Where:

- $lines—New broken up output.
- $output—Original output.

jcs:close() Function

The jcs:close() function closes a previously opened connection handle.

The function syntax is as follows:

```
expr jcs:close($connection);
```

Where $connection is a connection handle generated by a previously executed jcs:open() function.

jcs:dampen() Function

The jcs:dampen() function is used to limit the number of times a function is called in succession when using a script. If a specific function exceeds the maximum number of calls within a specified amount of time, the jcs:dampen() function returns false. If a function does not exceed the maximum number of calls within a specified amount of time, the function returns true.

The function syntax is as follows:

```
var $rc jcs:dampen($tag, max, frequency);
```

Where:

- $rc—Returned value based on the number of calls within a specified amount of time. If the number of calls exceeds the maximum number allowed, the value is false. If the number of calls is less than the maximum number allowed, the value is true.
- $tag—The function call measured.

- max—The maximum number of function calls allowed. This maximum number is based on the number of calls within a specified time interval (frequency).
- frequency—The time interval, in minutes,

The following example shows how the dampen() function can be used:

```
if (jcs:dampen('my-change', 3, 10)) {
    /* Code to handle situations where the my-change function */
    /*  is called more than 3 times within 10 minutes */
} else {
    /* Code to handle all other cases */
```

jcs:empty() Function

The jcs:empty() function returns true if a nodeset or string argument is empty.

The function syntax is as follows:

```
jcs:empty($set);
```

Where $set is the nodeset or string to test.

The following example shows how the jcs:empty() function can be used:

```
if (jcs:empty($set)); {
    /* Code to handle true value ($set is empty) */
}
```

jcs:execute() Function

The jcs:execute() function executes an RPC within the context of a specified connection handle. Any number of RPCs can be executed within the context of a specified connection handle until that connection handle is closed with the jcs:close() function.

The function syntax is as follows:

```
var $results = jcs:execute($connection, $rpc);
```

Where:

- $results—Results of the executed RPC. This $results varable is the same as the $results variable produced by the jcs:invoke() function.
- $connection—Connection handle generated by a previously executed jcs:open() function.
- $rpc—RPC to be executed. This $rpc varable is the same as the $rpc variable produced by the jcs:invoke() function.

jcs:first-of() Function

The jcs:first-of() function returns the first nonempty (non-null) item in a list. 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")
```

The following example selects the description of a logical interface if there is a logical interface description. If not, it selects the description of the (parent) physical interface if there is a physical interface description. If not, it selects the concatenation of the physical interface name with a " ." and the logical unit number.

```
<xsl:variable name="description"
      select="jcs:first-of(description, ../description, concat(../name, '.', name))"/>
```

jcs:getsecret() Function

The jcs:getsecret() function invokes a CLI prompt and waits for user input similar to the jcs:input() function. However, unlike jcs:input(), the user input will not be prompted/echoed back to the user, which is useful for entering user passwords. The user input is defined as a string for subsequent use. This function cannot be used with event scripts.

The function syntax is as follows

```
var $answer = jcs:getsecret("prompt ")
```

Where:

- prompt—CLI prompt.
- $answer—Returned user input.

The following example shows how to prompt for a password that will not be echoed back to the user:

```
var $password = jcs:getsecret("Enter password: ")
```

jcs:hostname() Function

The jcs:hostname() function returns the hostname associated with a given IP address.

The function syntax is as follows:

```
var $name = jcs:hostname($address);
```

Where:

- $name—Returned hostname.
- $address—IPv4 or IPv6 address.

jcs:input() Function

The jcs:input() function invokes a CLI prompt and waits for user input. The user input is defined as string for subsequent use. This function cannot be used with event scripts.

The function syntax is as follows

```
jcs:input("prompt" $answer)
```

Where:

- prompt—CLI prompt.
- $answer—Returned answer.

jcs:invoke() Function

The jcs:invoke() function invokes an RPC. It can be called with one argument, either a string containing a JUNOS XML or JUNOScript RPC method name or a tree containing an RPC. The result is the contents of the <rpc-reply> element, not the <rpc-reply> tag element itself.

In the following example, there is a test to see if the interface argument is included on the command line when the script is executed. If it is, the operational mode output of the show interfaces terse command is narrowed to include information about that interface only.

```
<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 an RPC without modifying the output:

```
<xsl:variable name="sw" select="jcs:invoke('get-software-information')"/>
```

jcs:open() Function

The jcs:open() function returns a connection handle that can be used to execute multiple RPCs using the jcs:execute() function and close the connection using the jcs:close() function.

The syntax is as follows:

```
var $connection = jcs:open($server, $username, $passphrase)
```

Where:

- $connection—Connection handle returned.
- $server—Domain name or IP address of the remote router. If you are opening a local connection, do not pass the $server value.
- $username—User's login name.
- $passphrase—User's login password.

The following example shows how the user bsmith with a passphrase password can open a connection with the server fivestar

```
$connection = jcs:open("fivestar", "bsmith", "password")
```

jcs:output() Function

The jcs:output() function allows you to make unformatted output text. It emits an <output> element. The text appears in the CLI.

jcs:parse-ip() Function

The jcs:parse-ip() function evaluates an IPv4 or IPv6 IP address and returns the following information:

- Hostname.
- Address family (inet4 for IPv4 or inet6 for IPv6).
- Prefix length.
- Network address.
- Netmask (for IPv4 address; left blank for IPv6 addresses).

The function syntax is as follows:

```
var $output = jcs:parse-ip(ipaddress/(prefix-length | netmask));
```

Where:

- $output—Results of the executed RPC.
- ipaddress—IPv4 or IPv6 address.
- prefix-length—Prefix length.
- netmask—Netmask.

jcs:printf() Function

The jcs:print() function emits formatted output text. The text appears in the CLI. Most standard printf formats are supported, in addition to some JUNOS software-specific ones. The %j1 operator emits the field only if the field was changed from the last time the function was run. The %jc operator capitalizes the first letter of the format output. The %jt{TAG} operator emits the tag if the field is not empty.

```
<xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ –> }s\n',
                        'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/>
```

jcs:progress() Function

The jcs:progress() function issues a progress message. It sends a progress message back to the client (CLI) containing its single argument.

```
<xsl:variable name="ignore" select="jcs:progress('Working...')"/>
```

jcs:regex() Function

The jcs:regex() function evaluates a regular expression against a given argument and returns any matches.

The function syntax is as follows:

```
var $pat = "(evaluation-pattern)";
var $output = ($pat, "expression")
```

Where:

- $pat—Specified pattern string to evaluate against the regular expression.
- $output—Matching values.

For example:

```
var $pat = "([0–9=) (:*)([a-z}*)";
var $a = jcs:regex($pat, "123:xyz");
```

Returns:

```
"123:xyz"
    ([0–9]+) returns "123"
    (:*) returns ":"
    ([a-z]*) returns "xyz"
```

For example:

```
var $pat = "([0–9=) (:*)([a-z}*)";
var $a = jcs:regex($pat, "r2d2");
```

Returns:

```
"2d"
    ([0–9]+) returns "2"
```

```
(:*) returns empty match
([a-z]*) returns "d"
```

jcs:sleep() Function

The jcs:sleep() function causes the script to sleep for a specified period. You can use this function to help determine how a routing component is working over time. To do this, write a script that issues a command, calls the jcs:sleep() function, and reissues the same command.

The *millis* argument is optional. For example, jcs:sleep(1) means 1 second and jcs:sleep(0, 10) means 10 milliseconds.

```
jcs:sleep(seconds <, millis>)
```

jcs:split() Function

The jcs:split() function splits a string into an array of substrings based on the regular expression pattern. If the optional argument limit is specified, only substrings up to the limit are returned.

```
jcs:split($pattern, $string, [$limit])
```

jcs:sysctl() Function

The jcs:sysctl() function returns a sysctl value as a string or integer.

The function syntax is as follows:

```
var $value = jcs:sysctl("sysctl-value", "s");
```

Where:

- $value—Returned string or integer value.
- sysctl-value—sysctl value to convert to a string or integer.

jcs:syslog() Function

The jcs:syslog() function logs messages to the syslog with a specified priority.

The function syntax is as follows:

```
expr jcs:syslog(priority, "message", <$goesto>, <$syslog>));
```

Where:

- priority—The priority given to the syslog message. The priority can be specified as a facility/severity pair, or it can be a numeric priority value based on the facility/severity pair. Table 22 on page 433 and Table 23 on page 433 show the facility and severity strings available and the corresponding numeric value for each string.

  The priority numeric value is calculated by multiplying the facility string value by 8 and adding the severity string. For example, if the facility/severity string pair is pfe/alert, the priority numeric value is 161 ((20 x 8) + 1).

- message—Message to add to the system log file.
- $goesto—(Optional) You can pass variable names as an argument to the function.
- $syslog—(Optional) You can pass the name of the system log file as an argument to the function.

The message added to the syslog file is specified within the [edit system syslog] hierarchy level of the router configuration file.

Table 22: 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
firewall	Firewall filtering system	19
ftp	FTP processes	11
interact	Commands executed by thye UI	23
pfe	Packet forwarding engine	20
user	User processes	1

Table 23: Severity String

Severity String	Description	Numeric Value
alert	Conditions that should be corrected immediately	1

Table 23: Severity String *(continued)*

Severity String	Description	Numeric Value
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

jcs:trace() Function

The jcs:trace() function issues a trace message, which is sent to the trace file.

Templates in the junos.xsl File

Named templates in the junos.xsl file use the jcs prefix to avoid conflicting with templates defined in commit scripts. The templates in the jcs namespace allow you to accomplish scripting tasks more easily. To use these templates in your scripts, you simply include <xsl:call-template name="*name*"> elements and pass in any required or optional parameters. The name attribute specifies the name of the called template.

The <xsl:template match="/"> template is an unnamed template that allows you to use shortened XPath expressions in your scripts.

These templates are discussed in more detail in the following sections:

- <jcs:edit-path> Template on page 434
- <jcs:emit-change> Template on page 435
- <jcs:emit-comment> Template on page 437
- <jcs:statement> Template on page 438

<jcs:edit-path> Template

The <jcs:edit> template generates an <edit-path> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The location of the configuration error is passed as dot into the <jcs:edit-path> template. This location defaults to " . " , the current position in the XML hierarchy. You can alter the default by including the select attribute of the dot parameter. The following example demonstrates how to call this 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 <xnm:warning> element 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

The <jcs:emit-change> template generates a <change> element, which results in a persistent change to the configuration.

This template includes the following optional parameters:

- <xsl:param name="content">—Allows you to include the content of the change, relative to dot.
- <xsl:param name="dot" select=".">—Allows you to indicate a location other than the current location in the XML hierarchy. The select attribute contains the current context " ." as a default value. If you want to change the current context, you can include the dot parameter and include a different XPath expression in the select attribute.
- <xsl:param name="message">—Allows you to include a warning message to be displayed by 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, include the dot parameter.
- <xsl:param name="name" select="name($dot)"/>—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 name of the element in $dot (which in turn defaults to " ." which is the current element).
- <xsl:param name="tag" select="'change'"/>—Allows you to specify the type of change to be generated. By default, the <jcs:emit-change> template generates a permanent change, as designated by the 'change' expression. To specify a transient change, you must include the tag parameter and include the 'transient-change' expression, as shown here:

  ```
  <xsl:with-param name="tag" select="'transient-change'"/>
  ```

The following example demonstrates how to call this 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 Intermediate System-to-Intermediate System (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:with-param name="dot" select="chassis"/>
```

<jcs:emit-comment> Template

The <jcs:emit-comment> template emits 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. 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:statement> Template

The <jcs:statement> template generates a <statement> element suitable for inclusion in an <xnm:error> or <xnm:warning> element. The parameter dot can be passed into the <jcs:statement> template if the error is not at the current position in the XML hierarchy. The following example demonstrates how to call this 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 “Imposing a Minimum MTU Setting” on page 195.

Chapter 29

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 JUNOScript 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] hierarchy level:

```
event-options {
    event-script {
        file filename;
        traceoptions {
            file filename <files number> <size size> <world-readable | no-world-readable>;
            flag flag;
        }
    }
}
```

This chapter discusses the following topics:

- Enabling an Event Script on page 439
- Executing an Event Script on page 440
- Declaring Arguments in Event Scripts on page 440
- Manually Converting a Script from XSLT to SLAX on page 441
- Manually Converting a Script from SLAX to XSLT on page 441
- Tracing Event Script Processing on page 442

Enabling an Event Script

After you write an event script, you must save it in the /var/db/scripts/event directory on the routing platform. Event scripts in the /var/db/scripts/event directory are ignored if they are not enabled. Only users in the superuser JUNOS login class can access and edit files in the /var/db/scripts/event directory.

If a platform has dual Routing Engines and you want the script to take effect on both Routing Engines, you must copy the script to the /var/db/scripts/event directory on

each Routing Engine. The commit synchronize command does not automatically copy the scripts from one Routing Engine directory into the other Routing Engine directory.

To enable an event script, include the file 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. When you enable a SLAX script, you must include the .slax extension in the filename. If you do not, the configuration commits, but the script cannot be executed. When you enable an XSLT script, the .xsl extension is optional but strongly recommended.

Executing an Event Script

Event scripts do not take effect when you issue a commit command. This command places the script into the system memory and enables it for execution on the system. Once the script is committed, you then execute the script to run it. An event script is executed in response to an event notification within an event policy. For more information on executing an event script, see “Executing Event Scripts in an Event Policy” on page 358

Declaring Arguments in Event Scripts

Arguments can be declared in an event script with XSLT (or SLAX) instructions in the script.

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";
   }
}
```

Example: Declaring Arguments

Declare two arguments named interface and protocol. Corresponding script parameters must be declared.

```
<xsl:param name="interface"/>
<xsl:param name="protocol"/>
```

Method: In the 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>
```

Manually Converting a Script from XSLT to SLAX

SLAX is a C-like alternative syntax to XSLT and can be viewed as a preprocessor for XSLT. Before the JUNOS software invokes the XSLT processor, the software converts SLAX constructs (such as if/then/else) to equivalent XSLT constructs (such as <xsl:choose> and <xsl:if>). For more information about SLAX, see "Understanding SLAX" on page 51.

To convert an XSLT script to SLAX, issue the request system scripts convert xslt-to-slax source /var/db/scripts/event/*filename* destination /var/db/scripts/event operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.xsl file remains in the /var/db/scripts/event directory and a new script called script1.slax is added to the /var/db/scripts/event directory.

To convert a script from SLAX to XSLT, see "Manually Converting a Script from SLAX to XSLT" on page 441.

Manually Converting a Script from SLAX to XSLT

To convert a SLAX script to XSLT, issue the request system scripts convert slax-to-xslt source /var/db/scripts/event/*filename* destination /var/db/scripts/event operational mode command:

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

The source script is the basis for a new script. The source script is not overwritten by the new script.

For example:

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

When you issue this command, the script1.slax file remains in the /var/db/scripts/event directory and a new script called script1.xsl is added to the /var/db/scripts/event directory.

To convert a script from XSLT to SLAX, see "Manually Converting a Script from XSLT to SLAX" on page 441.

Tracing Event Script Processing

Event script tracing operations track all 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 trace files is to log important events in a file called event-script.log located in the /var/log directory. When the file event-script.log reaches 128 kilobytes (KB), it is renamed and saved with a number 0 through 9 in ascending order appended to the end of the file. For example: event-script.log.0, then event-script.log.1, until there are 10 trace files. Then the oldest trace file (event-script.log.9) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

This chapter discusses the following topics:

- Minimum Configuration for Enabling and Viewing Traceoptions Output on page 442
- Configuring Traceoptions on page 444

Minimum Configuration for Enabling and Viewing Traceoptions Output

If no event script trace options are configured yet, the simplest way to view the trace output of an event script is to configure the output trace flag and issue the show log event-script.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]
   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]
   traceoptions flag output;
   ```

3. Issue the commit command:

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

4. Display the resulting trace messages recorded in the /var/log/event-script.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 event-script.log | last operational mode command:

```
[edit]
user@host# run show log event-script.log | last
```

Table 24 on page 443 summarizes useful filtering commands that display selected portions of the event-script.log file.

Table 24: Event Script Tracing Operational Mode Commands

Task	Command
Display logging data associated with all script processing.	show log event-script.log
Display script processing for only the most recent commit operation.	show log event-script.log | last
Display processing for script errors.	show log event-script.log | match error
Display script processing for a particular script.	show log event-script.log | match *script-name*

Example: Minimum Configuration for Enabling and Viewing Traceoptions Output

Trace the output of the event script file source-route:

```
system {
    event-options {
        event-script {
            file source-route;
            traceoptions flag output;
        }
    }
}
[edit]
user@host# commit
[edit]
user@host# run show log event-script.log | last
```

Configuring Traceoptions

You cannot change the /var/log directory in which trace files are located. However, you can customize the 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;
flag {
    all;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

These statements are described in the following sections:

- Configuring the Event Script Log Filename on page 444
- Configuring the Number and Size of Event Script Log Files on page 444
- Configuring the Event Script Trace Operations on page 445

Configuring the Event Script Log Filename

By default, the name of the file that records trace output is event-script.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 *filename*.0, then *filename*.1, and so on, until there are 10 trace files. Then the oldest trace file (*filename*.9) is overwritten.

You can configure the limits on the number and size of event script trace files by including the following statements at the [edit event-options event-script traceoptions file] or [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, *filename* is renamed *filename*.0, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename*.0 is renamed *filename*.1

and *filename* is renamed *filename*.0. This process repeats until there are 20 trace files. Then the oldest file (*filename*.19) is overwritten.

The number of files can be from 2 through 1000 files. The file size of each file can be from 10 KB through 1 gigabyte (GB).

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;
    events;
    input;
    offline;
    output;
    rpc;
    xslt;
}
```

Table 25 on page 445 describes the meaning of the event script tracing flags.

Table 25: 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 30

Event Script Examples

This chapter provides sample event scripts that run commands and customize output. This chapter includes the following topics:

- Limiting Event Script Output Based on a Specific Event Type on page 447

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 31

Summary of Event Script Configuration Statements

The following section explains each of the event script configuration statements. The statements are organized alphabetically.

event-script

Syntax

```
event-script {
    file filename
    traceoptions {
        file filename <files number> <size size>;
        flag flag;
    }
}
```

Hierarchy Level [edit event-options]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS event scripts, configure scripting mechanisms.

The statements are explained separately.

Usage Guidelines See “Configuring Event Scripts” on page 439.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

file

Syntax file *filename*

Hierarchy Level [edit event-options event-script]

Release Information Statement introduced in JUNOS Release 7.6.

Description For JUNOS 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.

Usage Guidelines See "Configuring Event Scripts" on page 439.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

traceoptions

Syntax

```
traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
}
```

Hierarchy Level [edit event-options event-script]

Release Information Statement introduced in JUNOS Release 7.6.

Description Define tracing operations for event scripts.

Default If you do not include this statement, no event-script-specific tracing operations are performed.

Options *filename*—Name of the file to receive the output of the tracing operation. All files are placed in the directory /var/log. By default, event script process tracing output is placed in the file event-script.log. If you include the file statement, you must specify a filename. To retain the default, you can specify event-script.log as the filename.

files *number*—(Optional) Maximum number of trace files. When a trace file named *trace-file* reaches its maximum size, it is renamed *trace-file*.0, then *trace-file*.1, and so on, until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum number of files, you also must specify a maximum file size with the size option and a filename.
Range: 2 through 1000
Default: 10 files

flag—Tracing operation to perform. To specify more than one tracing operation, include multiple flag statements. You can include the following flags:

- all—Log all operations
- events—Log important events
- input—Log 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

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 *trace-file*.0. When *trace-file* again reaches its maximum size, *trace-file*.0 is renamed *trace-file*.1 and *trace-file* is renamed *trace-file*.0. 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: *x*k to specify KB, *x*m to specify MB, or *x*g to specify GB
Range: 10 KB through 1 GB
Default: 128 KB

no-world-readable—Restrict file access to owner. This is the default.

world-readable—Enable unrestricted file access.

Usage Guidelines See "Tracing Event Script Processing" on page 442.

Required Privilege Level maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

Part 6
Index

- Index on page 455
- Index of Statements and Commands on page 463

Index

Symbols

#, comments in configuration statements.....xxxii
$
    regular expression operator
        event policy.....369
( ), in syntax descriptions.....xxxii
*
    regular expression operator
        event policy.....368
+
    regular expression operator
        event policy.....368
.
    regular expression operator
        event policy.....368
< >, in syntax descriptions.....xxxii
?
    regular expression operator
        event policy.....369
[ ], in configuration statements.....xxxii
^
    regular expression operator
        event policy.....369
{ }, in configuration statements.....xxxii
| (pipe)
    regular expression operator
        event policy.....369
| (pipe), in syntax descriptions.....xxxii

A

all
    commit script tracing flag.....175
    event policy tracing flag.....375
    event script tracing flag.....445
    op script tracing flag.....313
allow-transients statement.....179
    usage guidelines.....129
apply-macro statement.....180
    usage guidelines.....145
applying templates
    in SLAX.....58
    in XSLT.....19
archive-sites statement.....389
    usage guidelines.....353
archiving files.....353
arguments statement.....335, 390
    event policy, usage guidelines.....358
    event scripts, usage guidelines.....440
    op scripts, usage guidelines.....305
attributes
    for customized messages.....120
    in SLAX.....58
attributes-match statement.....390
    usage guidelines.....363

B

boilerplate
    for commit scripts.....92
    for event scripts.....419
    for op scripts.....285
braces, in configuration statements.....xxxii
brackets
    angle, in syntax descriptions.....xxxii
    square, in configuration statements.....xxxii

C

change element
    usage guidelines.....129, 145
command statement.....335
    usage guidelines.....304
commands statement.....391
    usage guidelines.....355, 358
comments
    in SLAX.....60
    in XSLT.....60
comments, in configuration statements.....xxxii
commit script example
    controlling CoS configuration.....260
    controlling default encapsulation.....217, 224
    controlling dual Routing Engines.....242
    controlling minimum MTU.....195
    controlling policy configuration.....246, 255
    decreasing manual configuration.....211, 214
    limiting number of ATM VCs.....208
    limiting number of interfaces.....199
    prohibiting configuration statements.....189

requiring configuration statements.....189
requiring internal clocking.....192
warning for deprecated statement.....197
commit scripts
all (tracing flag).....175
attributes for customized messages.....120
boilerplate.....92
commands for monitoring.....170
configuration statements.....179
deactivating.....163
deleting.....163
design considerations.....103
enabling.....163
error messages, generating.....109
example.....115
events (tracing flag).....175
examples.....104, 137, 189
extension functions.....41, 47
input (tracing flag).....175
macros.....145
making them optional.....163
multiple.....87
offline (tracing flag).....175
output (tracing flag).....175
output, displaying.....170
overview.....3, 79
persistent configuration changes.....129
example.....137
refresh from a different location.....168
remote sources
refreshing from.....166
specifying.....166
rpc (tracing flag).....175
superuser class, necessity of.....89
system log messages, generating.....109
example.....118
trace log files.....172
transient configuration changes.....129
example.....140
troubleshooting.....176
using multiple.....87
warning messages, generating.....109
example.....113
xslt (tracing flag).....175
commit statement.....181
usage guidelines.....163
concatenating arguments
SLAX.....56
configuration
event policy tracing flag.....375
generating changes
example.....137
persistent changes
example.....137
transient changes
example.....140
configuration mode commands
commit script.....170
context node.....25
controlling
CoS configuration
commit script example.....260
default encapsulation
commit script example.....217, 224
dual Routing Engines
commit script example.....242
minimum MTU
commit script example.....195
policy configuration
commit script example.....246, 255
conventions
text and syntax.....xxxi
converting
SLAX scripts to XSLT.....53, 169, 170, 310, 441
XSLT scripts to SLAX.....53, 169, 170, 310, 441
correlating events.....363
example configuration.....377, 385
curly braces, in configuration statements.....xxxii
customer support.....xxxiii
contacting JTAC.....xxxiii
customizing show command output
op script example.....320

D

dampening events
example configuration.....387
database
event policy tracing flag.....375
deactivating
scripts in the configuration.....163
decreasing manual configuration
commit script example.....211, 214
delaying file transfer.....353
deleting
scripts from the configuration.....163
description statement.....336
usage guidelines.....305, 307, 440
destination statement.....392
usage guidelines.....353, 355, 358
destinations statement.....393
usage guidelines.....353
direct-access statement.....181
documentation set
comments on.....xxxiii
dot node.....25
DTD
defined.....11

E

elements
in SLAX....56
else if statement in SLAX....53
else statement in SLAX....53
equals statement....393
usage guidelines....363
error messages, generating custom....109
example....115
event policy
all (tracing flag)....375
configuration (tracing flag)....375
configuration statements....389
configuring destinations....353
configuring file transfer delays....353
correlating events....363
example configuration....377, 385
dampening events
example configuration....387
database (tracing flag)....375
events (tracing flag)....375
example configurations....377
executing commands....355
executing op scripts....358
generating events....371
generating internal events
example configuration....386, 387
ignoring events....367
example configuration....385
KERNEL system log messages....373
LCC system log messages....373
nonstandard events....373
overview....4, 347
PFE system log messages....373
PIC system log messages....373
policy (tracing flag)....375
raising SNMP traps....372
regular expression filtering....368
regular expression operators....373
SCC system log messages....373
server (tracing flag)....375
syslogd(tracing flag)....375
SYSTEM system log messages....373
timer-events (tracing flag)....375
tracing operations....373
uploading event files....353
using regular expressions
example configuration....385
event scripts....416
arguments, declaring....440
boilerplate....419
configuration statements....449
configuring....439
enabling....439
example configurations....447
executing....440
extension functions....41, 47
overview....4, 415
programming....415
replacing....417
specifying memory location....416
superuser class, necessity of....417
trace log files....442
tracing flags
all....445
events....445
input....445
offline....445
output....445
rpc....445
xslt....445
usage guidelines....439
using....416
writing....419
event-options statement....394
usage guidelines....351
event-script statement....395, 449
usage guidelines....358
events
commit script tracing flag....175
event policy tracing flag....375
event script tracing flag....445
op script tracing flag....313
events statement....396
usage guidelines....363, 371
examples
commit scripts....189
event policy....377
event scripts....447
op scripts....315
execute-commands statement....397
usage guidelines....355, 358, 371, 372
executing operational-mode commands....355
expr statement in SLAX....56
expressions in SLAX....56

F

file statement....182, 337, 450
usage guidelines....163, 304, 373, 439
filename statement....397
usage guidelines....353
finding LSPs to multiple destinations
op script example....330
flag statement
usage guidelines....373
font conventions....xxxi
for-each statement in SLAX....53
functions
jcs:break-lines()....290, 426
jcs:close()....290, 426
jcs:dampen()....290, 426

jcs:empty()....291, 427
jcs:execute()....291, 427
jcs:first-of()....95, 292, 427
jcs:getsecret()....292, 428
jcs:hostname()....293, 428
jcs:input()....293, 429
jcs:invoke()....95, 293, 429
jcs:open()....294, 429
jcs:output()....95, 294, 430
jcs:parse-ip()....294, 430
jcs:printf()....96, 295, 431
jcs:progress()....95, 295, 431
jcs:regex()....295, 431
jcs:sleep()....96, 296, 432
jcs:split()....296, 432
jcs:sysctl()....296, 432
jcs:syslog()....297, 432
jcs:trace()....95, 298, 434

G

generate-event statement....398
usage guidelines....371
generating internal events....371
example configuration....386, 387

I

icons defined, notice....xxxi
if statement in SLAX....53
ignore statement....398
usage guidelines....367, 371, 372
ignoring events....367
example configuration....385
input
commit script tracing flag....175
event script tracing flag....445
op script tracing flag....313

J

jcs:break-lines() function....290, 426
jcs:close() function....290, 426
jcs:dampen() function....290, 426
jcs:edit-path template....97, 298, 434
jcs:emit-change template....97, 299, 435
jcs:emit-comment template....100, 301, 437
jcs:empty() function....291, 427
jcs:execute() function....291, 427
jcs:first-of() function....95, 292, 427
jcs:getsecret() function....292, 428
jcs:hostname() function....293, 428
jcs:input() function....293, 429
jcs:invoke() function....95, 293, 429
jcs:open() function....294, 429
jcs:output() function....95, 294, 430
jcs:parse-ip() function....294, 430
jcs:printf() function....96, 295, 431
jcs:progress() function....95, 295, 431
jcs:regex() function....295, 431
jcs:sleep() function....96, 296, 432
jcs:split() function....296, 432
jcs:statement template....100, 302, 438
jcs:sysctl() function....296, 432
jcs:syslog() function....297, 432
jcs:trace() function....95, 298, 434
JUNOS extension functions....41
JUNOS named templates....47
JUNOS XML API
introduction....9
JUNOS XML tags
notational conventions....10
junos.xsl file
functions....41, 94, 289, 425
importing....94, 289, 425
templates....47, 96, 298, 434
JUNOScript API
advantages of....11
introduction....9
JUNOScript RPCs
using in an op script....315
JUNOScript server....9
JUNOScript session
brief overview....12
JUNOScript tags
notational conventions....10

L

limiting number of ATM VCs
commit script example....208
limiting number of interfaces
commit script example....199
load-scripts-from-flash....282, 416
loading a base configuration
commit script example....263

M

macros
loading a base configuration....263
simplifying IGP configuration....228
simplifying interface configuration....232
simplifying IP address configuration....249
simplifying LDP configuration....220
simplifying MPLS LSP configuration....238
macros, commit scripts....145
advantages of....151
example configuration....153
manuals
comments on....xxxiii
match statement in SLAX....54

matches statement.....399
usage guidelines.....363, 368
multiple commit scripts.....87

N

named templates
in SLAX.....59
in XSLT.....19
no-world-readable statement
usage guidelines.....373
not statement.....399
usage guidelines.....363, 371, 372
notice icons defined.....xxxi
ns statement in SLAX.....55

O

offline
commit script tracing flag.....175
event script tracing flag.....445
op script tracing flag.....313
op script example
customizing show command output.....320
finding LSPs to multiple destinations.....330
restarting an FPC.....315
simplifying show command.....317
op scripts.....282
alias, defining.....304
arguments, declaring.....305
boilerplate.....285
configuration statements.....335
configuring.....303
enabling.....304
example configurations.....315
executing.....304
extension functions.....41, 47
help text, configuring.....307
master source
refreshing from.....308
specifying.....308
overview.....4, 281
programming.....281
refreshing.....308, 309
superuser class, necessity of.....283
trace log files.....311
tracing flags
all.....313
events.....313
input.....313
offline.....313
output.....313
rpc.....313
xslt.....313
usage guidelines.....303
using.....282
using JUNOScript RPCs.....315
writing.....285
Op scripts
specifying memory location.....282
op statement.....338
usage guidelines.....304
operational mode commands
displaying output fields in XML.....287, 421
for displaying commit script output.....170
using as an alternative to RPCs.....287, 422
with no XML equivalent.....287, 422
operators, regular expression
event policy.....368, 373
optional statement.....182
usage guidelines.....163
output
commit script tracing flag.....175
event script tracing flag.....445
op script tracing flag.....313
output-filename statement.....400
usage guidelines.....355, 358
output-format statement.....400
usage guidelines.....355, 358
overview
commit scripts.....3, 79
event policy.....4, 347
event scripts.....4, 415
op scripts.....4, 281
SLAX.....51
XML.....9
XSLT.....15

P

parameters
in SLAX.....57, 59
in XSLT.....20
parentheses, in syntax descriptions.....xxxii
persistent configuration changes
compared to transient changes.....130
example.....137
generating.....129
removing.....142
tags and attributes for.....143
policy
event policy tracing flag.....375
policy statement.....401
usage guidelines.....303, 351, 439
postinheritance, defined.....83
programming instructions, XSLT.....22
prohibiting configuration statements
commit script example.....189

R

raise-trap statement....402
usage guidelines....372
recursion, XSLT....25
refresh operation for commit scripts....166
refresh statement....183, 339
usage guidelines....166, 308
refresh-from statement....183, 339
usage guidelines....168, 309
regular expression operators
event policy....368, 373
remote source for commit scripts
refreshing from....166
specifying....166
request system scripts event-scripts reload....417
requiring configuration statements
commit script example....189
requiring internal clocking
commit script example....192
retry-count statement....402
rpc
commit script tracing flag....175
event script tracing flag....445
op script tracing flag....313
RPCs....287, 422
displaying output fields....287, 421
using in an op script....315

S

scripts statement....184, 340
usage guidelines....163
server 9
event policy tracing flag....375
See also JUNOScript server
session *See* JUNOScript session
simplifying
IGP configuration
commit script example....228
interface configuration
commit script example....232
IP address configuration
commit script example....249
LDP configuration
commit script example....220
MPLS LSP configuration
commit script example....238
show command
op script example....317
size statement
usage guidelines....373
SLAX....51
applying templates....58
attributes....58
benefits of....52
comments....60
converting to XSLT....53, 169, 170, 310, 441
elements....56
else if statement....53
else statement....53
expr statement....56
expressions....56
for-each statement....53
how SLAX works....52
if statement....53
if, else if, and else statements....53
introduction....51
jcs:split() function....296, 432
match statement....54
named templates....59
ns statement....55
overview....51
parameters....57
statement overviews....53
statement summaries....63
template parameters....59
using the XSL namespace....61
using XSLT elements....61
variables....57
version statement....55
SNMP traps
event policy....372
source statement....185, 341
usage guidelines....166, 308
starts-with statement....403
usage guidelines....363
statements in SLAX....53
superuser class
necessity of for commit scripts....89
necessity of for event scripts....417
necessity of for op scripts....283
support, technical *See* technical support
syntax conventions....xxxi
syslog element
usage guidelines....110
syslogd
event policy tracing flag....375
system log messages
in commit scripts....109
example....118
in event policies....373

T

tags *See* JUNOScript tags
tags for customized messages....120
technical support
contacting JTAC....xxxiii
templates
applying in SLAX....58
jcs:edit-path....97, 298, 434
jcs:emit-change....97, 299, 435

jcs:emit-comment....100, 301, 437
jcs:statement....100, 302, 438
parameters in SLAX....59
XSLT....18
templates, named
in SLAX....59
in XSLT....19
templates, unnamed
in XSLT....19
then statement....404
usage guidelines....353, 367
time-interval statement....405
usage guidelines....371
time-of-day statement....405
usage guidelines....371
timer-events
event policy tracing flag....375
traceoptions
for commit scripts....172
for event policy....373
for event scripts....442
for op scripts....311
traceoptions statement....186, 342, 406, 451
usage guidelines....172, 311, 373, 442
tracing event policy....373
tracing flags
commit scripts
all....175
events....175
input....175
offline....175
output....175
rpc....175
xslt....175
event policy
all....375
configuration....375
database....375
events....375
policy....375
server....375
syslogd....375
timer-events....375
event scripts
all....445
events....445
input....445
offline....445
output....445
rpc....445
xslt....445
op scripts
all....313
events....313
input....313
offline....313
output....313
rpc....313
xslt....313
tracing operations
commit scripts....172
event policy....373
event scripts....442
op scripts....311
transfer-delay statement....408
usage guidelines....353
transient configuration changes
compared to persistent changes....130
example....140
generating....129
removing....142
tags and attributes for....143
transient-change element
usage guidelines....129, 145
traps, SNMP....372
trigger statement....409
troubleshooting commit scripts....176

U

unnamed templates
in XSLT....19
upload statement....410
usage guidelines....353, 371, 372
uploading event files....353
user-name statement....411

V

variables
in SLAX....57
in XSLT....21
version statement in SLAX....55

W

warning for deprecated statement
commit script example....197
warning messages, generating custom....109
example....113
within statement....411
usage guidelines....363, 371, 372
world-readable statement
usage guidelines....373

X

XML
advantages of....11
overview....9
tags *See* JUNOS XML tags, JUNOScript tags

xnm:error element
usage guidelines....110
xnm:warning element
usage guidelines....110
XPath
functions....27
xsl
choose instruction....23
for-each instruction....23
if instruction....24
XSLT....27
attributes....27
context node....25
converting to SLAX....53, 169, 170, 310, 441
design considerations....103
dot node....25
elements....27
extension functions....41
flow of
operation....16, 52, 83, 101, 145, 283, 348
functions....27
introduction....15
jcs:break-lines() function....290, 426
jcs:close() function....290, 426
jcs:dampen() function....290, 426
jcs:edit-path template....97, 298, 434
jcs:emit-change template....97, 299, 435
jcs:emit-comment template....100, 301, 437
jcs:empty() function....291, 427
jcs:execute() function....291, 427
jcs:first-of() function....95, 292, 427
jcs:getsecret() function....292, 428
jcs:hostname() function....293, 428
jcs:input() function....293, 429
jcs:invoke() function....95, 293, 429
jcs:open() function....294, 429
jcs:output() function....95, 294, 430
jcs:parse-ip() function....294, 430
jcs:printf() function....96, 295, 431
jcs:progress() function....95, 295, 431
jcs:regex() function....295, 431
jcs:sleep() function....96, 296, 432
jcs:statement template....100, 302, 438
jcs:sysctl() function....296, 432
jcs:syslog() function....297, 432
jcs:trace() function....95, 298, 434
named templates....19, 47
namespace, in SLAX....61
overview....15
parameters....20
programming instructions....22
recursion....25
templates....18
unnamed templates....19
variables....21
XPath....17
xslt
commit script tracing flag....175
event script tracing flag....445
op script tracing flag....313

Index of Statements and Commands

A

allow-transients statement....179
apply-macro statement....180
archive-sites statement....389
arguments statement....335, 390
attributes-match statement....390

C

command statement....335
commands statement....391
commit statement....181

D

description statement....336
destination statement....392
destinations statement....393

E

equals statement....393
event-options statement....394
event-script statement....395, 449
events statement....396
execute-commands statement....397

F

file statement....182, 337, 450
filename statement....397

G

generate-event statement....398

I

ignore statement....398

L

load-scripts-from-flash....282, 416

M

matches statement....399

N

not statement....399

O

op statement....338
optional statement....182
output-filename statement....400
output-format statement....400

P

policy statement....401

R

raise-trap statement....402
refresh statement....183, 339
refresh-from statement....183, 339
request system scripts event-scripts reload....417
retry-count statement....402

S

scripts statement....184, 340
source statement....185, 341
starts-with statement....403

T

then statement....404
time-interval statement....405
time-of-day statement....405
traceoptions statement....186, 342, 406, 451
transfer-delay statement....408
trigger statement....409

U

upload statement....410

user-name statement.....411

W

within statement.....411