



JUNOS® Software

Configuration and Diagnostic Automation Guide

Release 10.0

Juniper Networks, Inc.

1194 North Mathilda Avenue
Sunnyvale, California 94089
USA

408-745-2000

www.juniper.net

Published: 2009-10-16

This product includes the Envoy SNMP Engine, developed by Epilogue Technology, an Integrated Systems Company. Copyright © 1986-1997, Epilogue Technology Corporation. All rights reserved. This program and its documentation were developed at private expense, and no part of them is in the public domain.

This product includes memory allocation software developed by Mark Moraes, copyright © 1988, 1989, 1993, University of Toronto.

This product includes FreeBSD software developed by the University of California, Berkeley, and its contributors. All of the documentation and software included in the 4.4BSD and 4.4BSD-Lite Releases is copyrighted by the Regents of the University of California. Copyright © 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994. The Regents of the University of California. All rights reserved.

GateD software copyright © 1995, the Regents of the University. All rights reserved. Gate Daemon was originated and developed through release 3.0 by Cornell University and its collaborators. Gated is based on Kirton's EGP, UC Berkeley's routing daemon (routed), and DCN's HELLO routing protocol. Development of Gated has been supported in part by the National Science Foundation. Portions of the GateD software copyright © 1988, Regents of the University of California. All rights reserved. Portions of the GateD software copyright © 1991, D. L. S. Associates.

This product includes software developed by Maker Communications, Inc., copyright © 1996, 1997, Maker Communications, Inc.

Juniper Networks, the Juniper Networks logo, JUNOS, NetScreen, ScreenOS, and Steel-Belted Radius are registered trademarks of Juniper Networks, Inc. in the United States and other countries. JUNOSe is a trademark of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Products made or sold by Juniper Networks or components thereof might be covered by one or more of the following patents that are owned by or licensed to Juniper Networks: U.S. Patent Nos. 5,473,599, 5,905,725, 5,909,440, 6,192,051, 6,333,650, 6,359,479, 6,406,312, 6,429,706, 6,459,579, 6,493,347, 6,538,518, 6,538,899, 6,552,918, 6,567,902, 6,578,186, and 6,590,785.

JUNOS® Software Configuration and Diagnostic Automation Guide

Release 10.0

Copyright © 2009, Juniper Networks, Inc.

All rights reserved. Printed in USA.

Writing: Michael Scruggs, Brenda Wilden

Editing: Sonia Saruba, Joanne McClintock, Nancy Kurahashi

Illustration: Faith Bradford

Cover Design: Edmonds Design

Revision History

October 2009—R1 JUNOS 10.0

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 NON-INFRINGEMENT. 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 présentes confirment leur volonté que cette convention de même que tous les documents y compris tout avis qui s'y rattache, soient rédigés 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 | Understanding SLAX | 27 |
| Chapter 6 | JUNOS Extension Functions and Templates in the jcs Namespace | 39 |
| Chapter 7 | Summary of XPath and XSLT Constructs | 59 |
| Chapter 8 | Summary of SLAX Statements | 91 |

Part 2

Commit Scripts

| | | |
|------------|---|-----|
| Chapter 9 | Commit Scripts Overview | 107 |
| Chapter 10 | Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message | 123 |
| Chapter 11 | Writing Commit Scripts That Generate a Persistent or Transient Configuration Change | 137 |
| Chapter 12 | Writing Commit Scripts That Create Custom Configuration Syntax with Macros | 153 |
| Chapter 13 | Configuring and Troubleshooting Commit Scripts | 167 |
| Chapter 14 | Commit Script Examples | 183 |
| Chapter 15 | Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts | 269 |
| Chapter 16 | Summary of Commit Script Configuration Statements | 277 |

Part 3

Operation (Op) Scripts

| | | |
|------------|---|-----|
| Chapter 17 | Op Scripts Overview | 289 |
| Chapter 18 | Writing Op Scripts | 291 |
| Chapter 19 | Configuring and Executing Op Scripts | 299 |
| Chapter 20 | Op Script Examples | 311 |
| Chapter 21 | Summary of Op Script Configuration Statements | 333 |

Part 4**Event Policy**

| | | |
|------------|--|-----|
| Chapter 22 | Event Policy Overview | 341 |
| Chapter 23 | Configuring Event Policy | 345 |
| Chapter 24 | Event Policy Examples | 371 |
| Chapter 25 | Summary of Event Policy Configuration Statements | 383 |

Part 5**Event Scripts**

| | | |
|------------|--|-----|
| Chapter 26 | Event Scripts Overview | 413 |
| Chapter 27 | Writing Event Scripts | 415 |
| Chapter 28 | Configuring Event Scripts | 423 |
| Chapter 29 | Event Script Examples | 435 |
| Chapter 30 | Summary of Event Script Configuration Statements | 437 |

Part 6**Index**

| | |
|----------------------------------|-----|
| Index | 447 |
| Index of Statements and Commands | 457 |

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 | xxix |
| | Merging a Snippet | xxx |
| | Documentation Conventions | xxx |
| | Documentation Feedback | xxxii |
| | Requesting Technical Support | xxxii |
| | Self-Help Online Tools and Resources | xxxiii |
| | Opening a Case with JTAC | xxxiii |
| Part 1 | Overview | |
| Chapter 1 | Overview of Configuration and Diagnostic Automation | 3 |
| | Commit Scripts Overview | 3 |
| | Op Scripts Overview | 4 |
| | Event Scripts Overview | 4 |
| | Event Policies Overview | 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 | 8 |
| Chapter 3 | Introduction to the JUNOS XML and JUNOScript APIs | 9 |
| | JUNOScript API and JUNOS XML API Overview | 9 |
| | XML Overview | 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

| | |
|---|----|
| XSLT Overview | 15 |
| XPath Overview | 17 |
| XSLT Templates Overview | 18 |
| Unnamed Templates | 19 |
| Named Templates | 19 |
| XSLT Parameter Passing Overview | 20 |
| XSLT Variables Overview | 21 |
| XSLT Programming Instructions Overview | 22 |
| <xsl:choose> Programming Instruction | 23 |
| <xsl:for-each> Programming Instruction | 23 |
| <xsl:if> Programming Instruction | 23 |
| Sample XSLT Programming Instructions and Pseudocode | 24 |
| XSLT Recursion Overview | 25 |
| XSLT Context (Dot) Overview | 25 |

Chapter 5 Understanding SLAX 27

| | |
|--|----|
| SLAX Overview | 27 |
| How SLAX Works | 28 |
| Converting Scripts Between SLAX and XSLT | 29 |
| SLAX Statements Overview | 29 |
| for-each Statement | 29 |
| if, else if, and else Statements | 29 |
| match Statement | 30 |
| ns Statement | 31 |
| version Statement | 31 |
| SLAX Elements Overview | 32 |
| XPATH Expressions Overview for SLAX | 32 |
| SLAX Variables and Parameters Overview | 33 |
| SLAX Element Attributes Overview | 33 |
| Template Matching Overview for SLAX | 34 |
| SLAX Parameter Passing Overview | 35 |
| Named Templates Overview for SLAX | 35 |
| SLAX Comments Overview | 36 |
| XSLT Elements Without SLAX Equivalents | 37 |

Chapter 6 JUNOS Extension Functions and Templates in the jcs Namespace 39

| | |
|--|----|
| JUNOS Extension Functions Overview | 39 |
| jcs:break-lines() Function | 40 |
| jcs:close() Function | 40 |
| jcs:dampen() Function | 41 |
| jcs:empty() Function | 41 |
| jcs:execute() Function | 42 |

| | |
|---|----|
| jcs:first-of() Function | 42 |
| jcs:get-input() Function | 43 |
| jcs:get-secret() Function | 43 |
| jcs:hostname() Function | 43 |
| jcs:invoke() Function | 44 |
| jcs:open() Function | 44 |
| jcs:output() Function | 45 |
| jcs:parse-ip() Function | 45 |
| jcs:printf() Function | 46 |
| jcs:progress() Function | 46 |
| jcs:regex() Function | 46 |
| jcs:sleep() Function | 47 |
| jcs:split() Function | 47 |
| jcs:sysctl() Function | 48 |
| jcs:syslog() Function | 49 |
| jcs:trace() Function | 50 |
| Importing the junos.xsl File | 51 |
| Templates in the junos.xsl File | 51 |
| <jcs:edit-path> Template | 51 |
| <jcs:emit-change> Template | 52 |
| <jcs:emit-comment> Template | 54 |
| <jcs:statement> Template | 55 |
| <xsl:template match = "/"> Template | 55 |

Chapter 7**Summary of XPath and XSLT Constructs****59**

| | |
|---|----|
| Summary of Standard XPath and XSLT Functions Referenced in This Guide | 60 |
| concat() | 60 |
| contains() | 60 |
| count() | 60 |
| last() | 61 |
| name() | 61 |
| not() | 61 |
| position() | 62 |
| starts-with() | 62 |
| string-length() | 62 |
| substring-after() | 63 |
| substring-before() | 63 |
| Summary of Standard XSLT Elements and Attributes Referenced in This Guide | 64 |
| <xsl:apply-templates> | 64 |
| <xsl:call-template> | 65 |
| <xsl:choose> | 65 |
| <xsl:comment> | 66 |
| <xsl:copy-of> | 66 |
| <xsl:element> | 66 |
| <xsl:for-each> | 67 |
| <xsl:if> | 67 |
| <xsl:import> | 68 |

| | |
|---|----|
| < xsl:otherwise > | 68 |
| < xsl:param > | 69 |
| < xsl:stylesheet > | 70 |
| < xsl:template > | 71 |
| < xsl:text > | 72 |
| < xsl:value-of > | 73 |
| < xsl:variable > | 73 |
| < xsl:when > | 74 |
| < xsl:with-param > | 75 |
| Summary of JUNOS XSLT Extension Functions | 76 |
| jcs:break-lines() | 76 |
| jcs:close() | 76 |
| jcs:dampen() | 77 |
| jcs:empty() | 77 |
| jcs:execute() | 78 |
| jcs:first-of() | 78 |
| jcs:get-input() | 78 |
| jcs:get-secret() | 79 |
| jcs:hostname() | 79 |
| jcs:invoke() | 80 |
| jcs:open() | 80 |
| jcs:output() | 81 |
| jcs:parse-ip() | 82 |
| jcs:printf() | 83 |
| jcs:progress() | 83 |
| jcs:regex() | 84 |
| jcs:sleep() | 84 |
| jcs:split() | 85 |
| jcs:sysctl() | 85 |
| jcs:syslog() | 86 |
| jcs:trace() | 86 |
| Summary of JUNOS Named XSLT Templates | 87 |
| < jcs:edit-path > | 87 |
| < jcs:emit-change > | 88 |
| < jcs:emit-comment > | 89 |
| < jcs:statement > | 89 |

Chapter 8

Summary of SLAX Statements

91

| | |
|-----------------|-----|
| apply-templates | 91 |
| call | 92 |
| else | 94 |
| for-each | 95 |
| if | 96 |
| match | 97 |
| mode | 98 |
| param | 99 |
| priority | 100 |
| template | 101 |
| var | 102 |

| | |
|---------------|-----|
| version | 103 |
| with | 104 |

Part 2

Commit Scripts

Chapter 9

Commit Scripts Overview **107**

| | |
|---|-----|
| Commit Scripts Overview | 107 |
| Advantages of Using Commit Scripts | 108 |
| How Commit Scripts Work | 109 |
| Commit Script Input | 110 |
| Commit Script Output | 111 |
| Commit Scripts and the JUNOS Software Commit Model | 112 |
| Standard Commit Model | 112 |
| Commit Model with Commit Scripts | 113 |
| Avoiding Potential Conflicts When Using Multiple Commit Scripts | 114 |
| Required Boilerplate for Commit Scripts | 115 |
| Design Considerations for Commit Scripts | 117 |
| Line-by-Line Explanation of Sample Commit Scripts | 119 |
| Applying a Change to SONET/SDH Interfaces | 119 |
| Applying a Change to ISO-Enabled Interfaces | 120 |

Chapter 10

Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message **123**

| | |
|---|-----|
| Overview of Generating Custom Warning, Error, and System Log Messages | 123 |
| Generating a Custom Warning, Error, or System Log Message | 124 |
| Tag Elements to Use When Generating Messages | 127 |
| Examples: Generating Custom Warning, Error, and System Log Messages | 129 |
| Example: Generating a Custom Warning Message | 129 |
| Verifying the Warning Message Generated by the Commit Script | 130 |
| Example: Generating a Custom Error Message | 132 |
| Verifying the Error Message Generated by the Commit Script | 133 |
| Example: Generating a Custom System Log Message | 135 |
| Verifying the System Log Message Generated by the Commit Script | 136 |

| | | |
|-------------------|--|------------|
| Chapter 11 | Writing Commit Scripts That Generate a Persistent or Transient Configuration Change | 137 |
| | Overview of Generating Persistent or Transient Configuration Changes | 137 |
| | Differences Between Persistent and Transient Changes | 137 |
| | Interaction of Configuration Changes and Configuration Groups | 140 |
| | Tag Elements and Templates for Generating Changes | 140 |
| | Generating a Persistent or Transient Change | 141 |
| | Removing a Persistent or Transient Change | 145 |
| | Tag Elements to Use When Generating Persistent and Transient Changes | 147 |
| | Examples: Generating Persistent and Transient Changes | 148 |
| | Example: Generating a Persistent Change | 148 |
| | Verifying the Persistent Change Generated by the Commit Script | 149 |
| | Example: Generating a Transient Change | 150 |
| | Verifying the Transient Change Generated by the Commit Script | 151 |
| Chapter 12 | Writing Commit Scripts That Create Custom Configuration Syntax with Macros | 153 |
| | Overview of Creating Custom Configuration Syntax with Macros | 153 |
| | How Macros Work | 153 |
| | Creating a Custom Syntax | 154 |
| | < data > Element | 155 |
| | Expanding the Custom Syntax | 156 |
| | Other Ways to Use Macros | 159 |
| | Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements | 159 |
| | Example: Creating Custom Configuration Syntax with Macros | 161 |
| | Verifying the Configuration Statements Generated by the Commit Script | 165 |
| Chapter 13 | Configuring and Troubleshooting Commit Scripts | 167 |
| | Implementing Commit Scripts | 168 |
| | Controlling Execution of Commit Scripts During Commit Operations | 168 |
| | Enabling Commit Scripts to Execute During Commit Operations | 169 |
| | Preventing Commit Scripts from Executing During Commit Operations | 170 |
| | Deactivating Commit Scripts | 170 |
| | Activating Commit Scripts | 171 |
| | Storing Commit Scripts in Flash Memory | 171 |
| | Overview of Updating Commit Scripts from a Remote Source | 171 |
| | Configuring the Master Source for a Commit Script | 173 |
| | Updating a Commit Script from the Master Source | 173 |
| | Updating a Commit Script from an Alternate Location | 174 |

| | |
|--|-----|
| Executing Large Commit Scripts | 174 |
| Converting a Commit Script from XSLT to SLAX | 175 |
| Converting a Commit Script from SLAX to XSLT | 175 |
| Displaying Commit Script Output | 176 |
| Tracing Commit Script Processing | 177 |
| Minimum Configuration for Tracing for Commit Script Operations | 177 |
| Example: Minimum Configuration for Enabling Traceoptions for Commit Scripts | 178 |
| Configuring Tracing of Commit Scripts | 179 |
| Configuring the Commit Script Log Filename | 179 |
| Configuring the Number and Size of Commit Script Log Files | 180 |
| Configuring Access to Commit Script Log Files | 180 |
| Configuring the Commit Script Trace Operations | 180 |
| Troubleshooting Commit Scripts | 181 |

Chapter 14

Commit Script Examples

183

| | |
|--|-----|
| Example: Requiring and Restricting Configuration Statements | 183 |
| Testing the ex-no-nukes Script | 186 |
| Example: Requiring Internal Clocking on T1 Interfaces | 187 |
| Testing the ex-clocking-error Script | 188 |
| Example: Imposing a Minimum MTU Setting | 189 |
| Testing the ex-so-mtu Script | 190 |
| Example: Limiting the Number of E1 Interfaces | 191 |
| Testing the ex-16-e1-limit Script | 192 |
| Example: Limiting the Number of ATM Virtual Circuits | 200 |
| Testing the ex-atm-vc-limit Script | 201 |
| Example: Controlling IS-IS and MPLS Interfaces | 203 |
| Testing the ex-iso Script | 205 |
| Example: Adding T1 Interfaces to a RIP Group | 206 |
| Testing the ex-rip-t1 Script | 208 |
| Example: Configuring a Default Encapsulation Type | 209 |
| Testing the ex-so-encap Script | 210 |
| Example: Controlling LDP Configuration | 212 |
| Testing the ex-ldp Script | 215 |
| Example: Adding a Final then accept Term to a Firewall | 216 |
| Testing the ex-add-accept Script | 218 |
| Example: Configuring an Interior Gateway Protocol on an Interface | 220 |
| Testing the ex-if-class Script | 222 |
| Example: Creating a Complex Configuration Based on a Simple Interface Configuration | 224 |
| Testing the ex-if-params Script | 228 |
| Example: Configuring Administrative Groups for LSPs | 230 |
| Testing the ex-lsp-admin Script | 232 |
| Example: Configuring Dual Routing Engines | 234 |
| Testing the ex-dual-re and ex-dual-re2 Scripts | 237 |
| Example: Preventing Import of the Full Routing Table | 238 |
| Testing the ex-import Script | 240 |
| Example: Automatically Configuring Logical Interfaces and IP Addresses | 241 |
| Testing the ex-atm-logical Script | 246 |

| | |
|---|-----|
| Example: Prepending a Global Policy | 247 |
| Testing the ex-bgp-global-import Script | 249 |
| Example: Assigning a Classifier | 252 |
| Testing the ex-classifier Script | 253 |
| Example: Loading a Base Configuration | 255 |
| Testing the config-system Script | 267 |

Chapter 15

| | |
|--|------------|
| Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts | 269 |
|--|------------|

| | |
|-----------------------------------|-----|
| < change > (XSLT) | 270 |
| < syslog > (JUNOS XML) | 271 |
| < transient-change > (XSLT) | 272 |
| < xnm:error > (JUNOS XML) | 273 |
| < xnm:warning > (JUNOS XML) | 275 |

Chapter 16

| | |
|--|------------|
| Summary of Commit Script Configuration Statements | 277 |
|--|------------|

| | |
|--|-----|
| allow-transients | 277 |
| apply-macro | 278 |
| commit | 279 |
| direct-access | 279 |
| file (Commit Scripts) | 280 |
| optional | 280 |
| refresh (Commit Scripts) | 281 |
| refresh-from (Commit Scripts) | 281 |
| scripts | 282 |
| source (Commit Scripts) | 283 |
| traceoptions (Commit and Op Scripts) | 284 |

Part 3

| |
|-------------------------------|
| Operation (Op) Scripts |
|-------------------------------|

Chapter 17

| | |
|----------------------------|------------|
| Op Scripts Overview | 289 |
|----------------------------|------------|

| | |
|--------------------------------------|-----|
| Op Script Programming Overview | 289 |
| How Op Scripts Work | 289 |

| | | |
|-------------------|---|------------|
| Chapter 18 | Writing Op Scripts | 291 |
| | Required Boilerplate for Op Scripts | 291 |
| | Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation | 293 |
| | Using RPCs and Operational Mode Commands in Op Scripts | 294 |
| | Using RPCs in Op Scripts | 294 |
| | Using Operational Mode Commands in Op Scripts | 294 |
| | Declaring Arguments in Op Scripts | 295 |
| | Example: Declaring Arguments | 297 |
| | Configuring Help Text for Op Scripts | 298 |
| | Examples: Configuring Help Text for Op Scripts | 298 |
| Chapter 19 | Configuring and Executing Op Scripts | 299 |
| | Implementing Op Scripts | 300 |
| | Enabling an Op Script and Defining a Script Alias | 300 |
| | Executing an Op Script | 301 |
| | Executing an Op Script by Issuing the op Command | 301 |
| | Executing an Op Script at Login | 302 |
| | Storing Op Scripts in Flash Memory | 302 |
| | Specifying a Master Source for an Op Script | 302 |
| | Updating an Op Script from the Master Source | 303 |
| | Updating an Op Script from an Alternate Location | 304 |
| | Converting an Op Script from XSLT to SLAX | 304 |
| | Converting an Op Script from SLAX to XSLT | 305 |
| | Tracing Op Script Processing | 305 |
| | Minimum Configuration for Enabling Traceoptions for Op Scripts | 306 |
| | Example: Minimum Configuration for Enabling Traceoptions for Op Scripts | 307 |
| | Configuring Tracing of Op Scripts | 307 |
| | Configuring the Op Script Log Filename | 307 |
| | Configuring the Number and Size of Op Script Log Files | 308 |
| | Configuring Access to Op Script Log Files | 308 |
| | Configuring the Op Script Trace Operations | 308 |
| Chapter 20 | Op Script Examples | 311 |
| | Example: Restarting an FPC in an Op Script | 311 |
| | Testing the ex-fpc Script | 312 |
| | Example: Displaying DNS Hostname Information in an Op Script | 313 |
| | Testing the ex-hostname Script | 316 |
| | Example: Customizing Output of the show interfaces terse Command in an Op Script | 316 |
| | Line-by-Line Explanation of the Script | 319 |
| | Testing the ex-interface Script | 325 |

| | |
|--|-----|
| Example: Finding LSPs to Multiple Destinations in an Op Script | 326 |
| Testing the ex-lsp Script | 329 |
| Example: Importing and Exporting Files in an Op Script | 330 |
| Exporting Files to a Remote Server | 330 |
| Importing Files from a Remote Server | 330 |

| | | |
|-------------------|--|------------|
| Chapter 21 | Summary of Op Script Configuration Statements | 333 |
|-------------------|--|------------|

| | |
|---------------------------------|-----|
| arguments | 333 |
| command | 334 |
| description | 334 |
| file (Op Scripts) | 335 |
| op | 336 |
| refresh (Op Scripts) | 337 |
| refresh-from (Op Scripts) | 337 |
| scripts | 338 |
| source (Op Scripts) | 338 |
| traceoptions | 338 |

| | |
|---------------|---------------------|
| Part 4 | Event Policy |
|---------------|---------------------|

| | | |
|-------------------|------------------------------|------------|
| Chapter 22 | Event Policy Overview | 341 |
|-------------------|------------------------------|------------|

| | |
|---|-----|
| Event Notifications and Policies Overview | 341 |
| How Event Policies Work | 342 |

| | | |
|-------------------|---------------------------------|------------|
| Chapter 23 | Configuring Event Policy | 345 |
|-------------------|---------------------------------|------------|

| | |
|---|-----|
| Using Correlated Events to Trigger an Event Policy | 347 |
| Representing the Correlating Event in an Event Policy | 349 |
| Triggering an Event Policy Based on Event Count | 350 |
| Using Regular Expressions to Refine the Set of Events That Trigger a Policy | 350 |
| Generating Internal Events to Trigger Event Policies | 351 |
| Using Nonstandard System Log Messages to Trigger Event Policies | 352 |
| Defining Destinations for File Archiving by Event Policies | 353 |
| Configuring an Event Policy to Upload Files | 354 |
| Configuring the Delay Before Files Are Uploaded by an Event Policy | 355 |
| Configuring an Event Policy to Retry the File Upload Action | 356 |
| Configuring an Event Policy to Execute Operational Mode Commands | 357 |
| Executing Event Scripts in an Event Policy | 360 |
| Configuring Event Policies to Ignore an Event | 365 |
| Changing the User Privilege Level for an Event Policy Action | 366 |
| Configuring Event Policies to Raise SNMP Traps | 366 |
| Tracing Event Policy Processing | 367 |
| Configuring the Event Policy Log Filename | 368 |
| Configuring the Number and Size of Event Policy Log Files | 368 |

| | |
|---|-----|
| Configuring Access to the Log File | 368 |
| Configuring a Regular Expression for Lines to Be Logged | 369 |
| Configuring the Trace Operations | 369 |

Chapter 24**Event Policy Examples****371**

| | |
|---|-----|
| Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval | 371 |
| Examples: Assigning a Transfer Delay to an Event Policy Action | 372 |
| Example: Representing the Correlating Event in an Event Policy | 374 |
| Example: Associating an Optional User with an Event Policy Action | 374 |
| Examples: Retrying the File Upload Action | 375 |
| Examples: Triggering a Policy Based on Event Count | 377 |
| Example: Ignoring Events Based on Receipt of Other Events | 378 |
| Example: Correlating Events Based on Event Attributes | 379 |
| Controlling Event Policy Using a Regular Expression | 380 |
| Example: Generating an Internal Event Every Hour | 380 |
| Example: Generating an Internal Event at Midnight | 381 |
| Example: Raising an SNMP Trap in Response to an Event | 381 |
| Example: Using Nonstandard System Log Messages to Trigger an Event Policy | 381 |

Chapter 25**Summary of Event Policy Configuration Statements****383**

| | |
|---|-----|
| archive-sites | 383 |
| arguments | 384 |
| attributes-match | 384 |
| commands | 385 |
| destination | 386 |
| destinations | 387 |
| equals | 387 |
| event-options | 388 |
| event-script | 390 |
| events | 391 |
| events (Associating Events with a Policy) | 392 |
| events (Correlating Events with Each Other) | 392 |
| execute-commands | 393 |
| generate-event | 394 |
| ignore | 394 |
| matches | 395 |
| not | 396 |
| output-filename | 396 |
| output-format | 397 |
| policy | 398 |
| raise-trap | 400 |
| retry-count | 400 |
| starts-with | 401 |
| then | 402 |
| time-interval | 404 |
| time-of-day | 404 |

| | |
|----------------------|-----|
| traceoptions | 405 |
| transfer-delay | 407 |
| trigger | 408 |
| upload | 409 |
| user-name | 410 |
| within | 410 |

Part 5

Event Scripts

Chapter 26

Event Scripts Overview **413**

| | |
|---|-----|
| Event Script Programming Overview | 413 |
| How Event Scripts Work | 413 |

Chapter 27

Writing Event Scripts **415**

| | |
|--|-----|
| Required Boilerplate for Event Scripts | 415 |
| Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation | 417 |
| Using RPCs and Operational Mode Commands in Event Scripts | 417 |
| Using RPCs in Event Scripts | 418 |
| Using Operational Mode Commands in Event Scripts | 419 |
| Capturing and Using Event Details and Remote Execution Details In Event Scripts | 420 |

Chapter 28

Configuring Event Scripts **423**

| | |
|---|-----|
| Implementing Event Scripts | 424 |
| Installing Event Scripts on a Router | 424 |
| Replacing an Event Script | 424 |
| Enabling an Event Script | 425 |
| Executing an Event Script | 426 |
| Storing Event Scripts in Flash Memory | 426 |
| Specifying a Master Source for an Event Script | 426 |
| Updating an Event Script from the Master Source | 427 |
| Updating an Event Script from an Alternate Location | 428 |
| Converting an Event Script from XSLT to SLAX | 428 |
| Converting an Event Script from SLAX to XSLT | 429 |
| Tracing Event Script Processing | 429 |
| Minimum Configuration for Enabling Traceoptions for Event Scripts | 430 |
| Example: Minimum Configuration for Enabling Traceoptions for Event Scripts | 431 |
| Configuring Tracing of Event Scripts | 431 |
| Configuring the Event Script Log Filename | 431 |
| Configuring the Number and Size of Event Script Log Files | 432 |
| Configuring Access to Event Script Log Files | 432 |
| Configuring the Event Script Trace Operations | 432 |

| | | |
|-------------------|--|------------|
| Chapter 29 | Event Script Examples | 435 |
| | Example: Limiting Event Script Output Based on a Specific Event Type | 435 |
| Chapter 30 | Summary of Event Script Configuration Statements | 437 |
| | event-script | 437 |
| | file | 438 |
| | refresh | 439 |
| | refresh-from | 439 |
| | remote-execution | 440 |
| | source | 441 |
| | traceoptions | 442 |
| Part 6 | Index | |
| | Index | 447 |
| | Index of Statements and Commands | 457 |

List of Figures

Part 1

Overview

| | | |
|-----------|--|----|
| Chapter 4 | Understanding XSLT | 15 |
| | Figure 1: Flow of XSLT Script Through the XSLT Engine | 16 |
| Chapter 5 | Understanding SLAX | 27 |
| | Figure 2: SLAX Script Input and Output | 28 |
| Chapter 6 | JUNOS Extension Functions and Templates in the jcs Namespace | 39 |
| | Figure 3: Commit Script Input and Output | 56 |

Part 2

Commit Scripts

| | | |
|------------|--|-----|
| Chapter 9 | Commit Scripts Overview | 107 |
| | Figure 4: Commit Script Input and Output | 110 |
| | Figure 5: Standard Commit Model | 112 |
| | Figure 6: Commit Model with Commit Scripts Added | 113 |
| | Figure 7: Configuration Evaluation by Multiple Commit Scripts | 115 |
| Chapter 12 | Writing Commit Scripts That Create Custom Configuration Syntax with Macros | 153 |
| | Figure 8: Macro Input and Output | 153 |
| | Figure 9: Sample Macro and Corresponding JUNOS CLI Expansion | 161 |

Part 3

Operation (Op) Scripts

| | | |
|------------|---|-----|
| Chapter 17 | Op Scripts Overview | 289 |
| | Figure 10: Op Script Input and Output | 290 |

Part 4

Event Policy

| | | |
|------------|--|-----|
| Chapter 22 | Event Policy Overview | 341 |
| | Figure 11: Interaction of eventd Process with Other JUNOS Software Processes | 341 |

List of Tables

| | | |
|---------------|--|--------------|
| | About This Guide | xxvii |
| | Table 1: Notice Icons | xxxi |
| | Table 2: Text and Syntax Conventions | xxxi |
| Part 1 | Overview | |
| Chapter 4 | Understanding XSLT | 15 |
| | Table 3: XSLT Concepts | 16 |
| | Table 4: Examples and Pseudocode for XSLT Variable Declaration | 22 |
| | Table 5: Examples and Pseudocode for XSLT Programming Instructions | 24 |
| Chapter 6 | JUNOS Extension Functions and Templates in the jcs Namespace | 39 |
| | Table 6: Facility Strings | 49 |
| | Table 7: Severity Strings | 50 |
| Part 2 | Commit Scripts | |
| Chapter 10 | Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message | 123 |
| | Table 8: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages | 127 |
| Chapter 11 | Writing Commit Scripts That Generate a Persistent or Transient Configuration Change | 137 |
| | Table 9: Differences Between Persistent and Transient Changes | 138 |
| | Table 10: Tags and Attributes for Creating Configuration Changes | 147 |
| Chapter 13 | Configuring and Troubleshooting Commit Scripts | 167 |
| | Table 11: Commit Script Configuration and Operational Mode Commands | 176 |
| | Table 12: Commit Script Tracing Operational Mode Commands | 178 |
| | Table 13: Commit Script Tracing Flags | 181 |
| | Table 14: Troubleshooting Commit Scripts | 181 |
| Part 3 | Operation (Op) Scripts | |
| Chapter 19 | Configuring and Executing Op Scripts | 299 |
| | Table 15: Op Script Tracing Operational Mode Commands | 306 |
| | Table 16: Op Script Tracing Flags | 309 |
| Part 4 | Event Policy | |
| Chapter 23 | Configuring Event Policy | 345 |

| | | |
|-------------------|--|------------|
| | Table 17: Regular Expression Operators for the matches Statement | 351 |
| | Table 18: Event ID by System Log Message Origin | 352 |
| | Table 19: Event Policy Tracing Flags | 369 |
| Chapter 24 | Event Policy Examples | 371 |
| | Table 20: Event Count Triggers Policy | 377 |
| Part 5 | Event Scripts | |
| Chapter 28 | Configuring Event Scripts | 423 |
| | Table 21: Event Script Tracing Operational Mode Commands | 430 |
| | Table 22: Event Script Tracing Flags | 433 |

About This Guide

This preface provides the following guidelines for using the *JUNOS® 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 xxx
- Documentation Feedback on page xxxii
- Requesting Technical Support on page xxxii

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 release notes differs from the information in the documentation, follow the *JUNOS Software Release Notes*.

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

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

Objectives

This guide provides an overview, instructions for using, and examples of JUNOS automation and the self-diagnosis features of the JUNOS Software. JUNOS automation scripts, which include commit scripts, operation scripts, and event scripts, are based on Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) 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: For additional information about 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, 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 1 on page xxxi defines notice icons used in this guide.

Table 1: Notice Icons





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

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

Table 2: Text and Syntax Conventions

| Convention | Description | Examples |
|------------------------------|---|--|
| Bold text like this | Represents text that you type. | To enter configuration mode, type the <code>configure</code> command: user@host> configure |
| Fixed-width text like this | Represents output that appears on the terminal screen. | user@host> show chassis alarms No alarms currently active |
| <i>Italic text like this</i> | <ul style="list-style-type: none"> Introduces important new terms. Identifies book names. Identifies RFC and Internet draft titles. | <ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>JUNOS System Basics Configuration Guide</i> RFC 1997, <i>BGP Communities Attribute</i> |
| <i>Italic text like this</i> | 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 <i>domain-name</i> |
| Plain text like this | Represents names of configuration statements, commands, files, and directories; IP addresses; configuration hierarchy levels; or labels on routing platform components. | <ul style="list-style-type: none"> 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 <i>metric</i> >; |

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

| Convention | Description | Examples |
|--------------------------------|--|--|
| (pipe symbol) | Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity. | broadcast multicast (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] |
| Indentation 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. | <ul style="list-style-type: none">■ In the Logical Interfaces box, select All Interfaces.■ To cancel the configuration, click Cancel. |
| > (bold right angle bracket) | Separates levels in a hierarchy of J-Web selections. | In the configuration editor hierarchy, select Protocols > Ospf . |

Documentation Feedback

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

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

Requesting Technical Support

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

contract, or are covered under warranty, and need 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: <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
- Understanding SLAX on page 27
- JUNOS Extension Functions and Templates in the jcs Namespace on page 39
- Summary of XPath and XSLT Constructs on page 59
- Summary of SLAX Statements on page 91

Chapter 1

Overview of Configuration and Diagnostic Automation

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

This chapter discusses the following topics:

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

Commit Scripts Overview

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 more detailed information, see “Commit Scripts Overview” on page 107.

Op Scripts Overview

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 more detailed information, see “Op Script Programming Overview” on page 289.

Event Scripts Overview

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 more detailed information, see “Event Script Programming Overview” on page 413.

Event Policies Overview

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 more detailed information, see “Event Notifications and Policies Overview” on page 341.

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

```
[edit event-options]
destinations {
  destination-name {
    archive-sites {
      url <password password>;
    }
    transfer-delay seconds;
  }
}
event-scripts {
  file filename {
    refresh;
    refresh-from url;
    remote-execution {
      remote-hostname {
        passphrase user-password;
        username user-login;
      }
    }
    source url;
  }
  refresh;
  refresh-from url;
  traceoptions {
    file <filename> <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
  }
}
generate-event event-name {
  time-interval seconds;
  time-of-day hh:mm:ss;
}
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 ];
  within seconds {
    events [ events ];
    not events [ events ];
    trigger (on | after | until) event-count;
  }
  then {
    event-script filename {
      arguments {
        argument-name argument-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 (filename | committed) destination destination-name {
        retry-count number retry-interval seconds;
        transfer-delay seconds;
        user-name username;
    }
}
}
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*.

```
[edit system]
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> <world-readable | no-world-readable>;
      flag flag;
      no-remote-trace;
    }
  }
}
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;
  source url;
  traceoptions {
    file <filename> <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
  }
}
}
```

Chapter 3

Introduction to the JUNOS XML and JUNOScript APIs

This chapter discusses the following topics:

- JUNOScript API and JUNOS XML API Overview on page 9
- XML Overview on page 10
- Advantages of Using the JUNOScript and JUNOS XML APIs on page 11
- Overview of a JUNOScript Session on page 12

JUNOScript API and JUNOS XML API Overview

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.

XML Overview

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

For more details about XML, see *A Technical Introduction to XML* at <http://www.xml.com/pub/a/98/10/guide0.html> and the additional reference material at the <http://www.xml.com> site. The official XML specification from the World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0*, is available at <http://www.w3.org/TR/REC-xml>.

The following sections discuss XML and JUNOScript tag elements:

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

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

This chapter discusses the following topics:

- XSLT Overview on page 15
- XPath Overview on page 17
- XSLT Templates Overview on page 18
- XSLT Parameter Passing Overview on page 20
- XSLT Variables Overview on page 21
- XSLT Programming Instructions Overview on page 22
- XSLT Recursion Overview on page 25
- XSLT Context (Dot) Overview on page 25

XSLT Overview

Commit scripts, op scripts, and event scripts can be 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) and accessible 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,

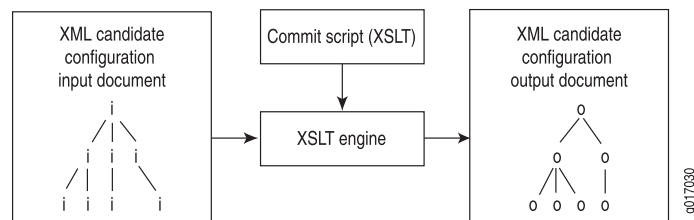
such as opening random Transmission Control Protocol (TCP) ports, forking numerous processes, or sending e-mail. The only action available in XSLT is to generate XML, and the XML is interpreted by the UI according to fixed semantics. An XSLT script can output only XML data, which is directly processed 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 3 on page 16.

Table 3: XSLT Concepts

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

XPath Overview

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 “Required Boilerplate for Commit Scripts” on page 115.

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 Constructs” on page 59. We also recommend consulting a comprehensive XPath reference guide. XPath is fully described in the W3C specification at <http://w3c.org/TR/xpath>.

XSLT Templates Overview

An XSLT script consists of one or more sets of rules called *templates*. Each template 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 **<div class="author">** 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 “**<xsl:template match = "/" > Template**” on page 55.

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 “Example: Requiring and Restricting Configuration Statements” on page 183.

XSLT Parameter Passing Overview

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>
```

XSLT Variables Overview

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 4 on page 22 provides examples of XSLT variable declarations along with pseudocode explanations.

Table 4: Examples and Pseudocode for XSLT Variable Declaration

| Variable Declaration | Pseudocode Explanation |
|---|---|
| <code><<xsl:variable> name="mpls" select="protocols/mpls"/></code> | Assigns the [edit protocols mpls] hierarchy level to the variable named \$mpls. |
| <code><xsl:variable> name="color" select="data[name = 'color']/value"/></code> | 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 159. |

XSLT Programming Instructions Overview

XSLT has a number of traditional programming instructions. Their form tends to be verbose, because their syntax is built from XML elements. For summaries of all XSLT programming instructions used in this guide, see “Summary of XPath and XSLT Constructs” on page 59.

The XSLT programming instructions most commonly used in commit, op, and event scripts are described in the following sections:

- `<xsl:choose>` Programming Instruction on page 23
- `<xsl:for-each>` Programming Instruction on page 23

- `<xsl:if>` Programming Instruction on page 23
- Sample XSLT Programming Instructions and Pseudocode on page 24

`<xsl:choose>` Programming Instruction

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>` Programming Instruction

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>` Programming Instruction

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

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

</xsl:if>

Sample XSLT Programming Instructions and Pseudocode

Table 5 on page 24 presents examples that use several XSLT programming instructions along with pseudocode explanations.

Table 5: Examples and Pseudocode for XSLT Programming Instructions

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

XSLT Recursion Overview

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

To avoid infinite recursion and excessive consumption of system resources, the JUNOS 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 < $max">
    <xsl:call-template name="count">
      <xsl:with-param name="cur" select="$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.

XSLT Context (Dot) Overview

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

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

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

Chapter 5

Understanding SLAX

This chapter discusses the following topics:

- SLAX Overview on page 27
- How SLAX Works on page 28
- Converting Scripts Between SLAX and XSLT on page 29
- SLAX Statements Overview on page 29
- SLAX Elements Overview on page 32
- XPATH Expressions Overview for SLAX on page 32
- SLAX Variables and Parameters Overview on page 33
- SLAX Element Attributes Overview on page 33
- Template Matching Overview for SLAX on page 34
- SLAX Parameter Passing Overview on page 35
- Named Templates Overview for SLAX on page 35
- SLAX Comments Overview on page 36
- XSLT Elements Without SLAX Equivalents on page 37

SLAX 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 183 and “Op Script Examples” on page 311.

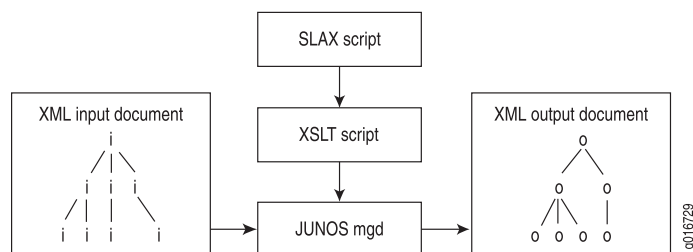
How SLAX Works

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 internally translates 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 translation, the XSLT transformation engine—which, for the JUNOS Software, is the JUNOS management (mgd) process—is invoked.

Figure 2 on page 28 shows the flow of SLAX script input and output.

Figure 2: SLAX Script Input and Output



Converting Scripts Between SLAX and XSLT

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 “Converting a Commit Script from XSLT to SLAX” on page 175 or “Converting an Op Script from XSLT to SLAX” on page 304 and “Converting a Commit Script from SLAX to XSLT” on page 175 or “Converting an Op Script from SLAX to XSLT” on page 305.

SLAX Statements Overview

This section lists some commonly used SLAX statements, with brief examples and XSLT equivalents. For a complete list of SLAX statements, see “Summary of SLAX Statements” on page 91.

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 style sheet or at the beginning of any block.

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

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

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

The XSLT equivalent:

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

version Statement

All SLAX style sheets must begin with a **version** statement, which specifies the version number for the SLAX language. 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">
```

SLAX Elements Overview

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.

XPATH Expressions Overview for SLAX

XPath expressions can appear either as the contents of an XML element or as the contents of an `expr` (expression) statement. In either case, the value is translated to 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 XSLT equivalent:

```

<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>

```

SLAX Variables and Parameters Overview

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.')."/>

```

SLAX Element Attributes Overview

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}}"/>
```

Template Matching Overview for SLAX

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>
```

SLAX Parameter Passing Overview

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 Overview for SLAX

The named template definition consists of the **template** keyword, the template name, a set of parameters, and a braces-delimited block of code. Parameter declarations can be inline and consist of the parameter name, and 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>

```

SLAX Comments Overview

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
 */ -->

```

XSLT Elements Without SLAX Equivalents

Some XSLT elements are not directly translated into SLAX statements. Some examples of XSLT elements for which there are no SLAX equivalents 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 6

JUNOS Extension Functions and Templates in the jcs Namespace

When you write scripts, you can use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided by the JUNOS Software. These tools include extension functions that accomplish scripting tasks more easily, and the import file called `junos.xsl`, which includes named templates that make scripts easier to read and write. The extension functions and templates use the `jcs:` prefix to avoid conflicts with standard XSLT functions and user-defined templates. This chapter discusses the extension functions and templates in detail in the following sections:

- JUNOS Extension Functions Overview on page 39
- Importing the `junos.xsl` File on page 51

JUNOS Extension Functions Overview

The JUNOS extension functions are used in commit, op and event scripts to accomplish scripting tasks more easily. Extension functions allow you to perform operations that are difficult or impossible to perform in XPath. The functions use the `jcs` prefix to avoid conflicting with standard XSLT functions. To use functions in the `jcs` namespace in your scripts, you must map to the `jcs` namespace in your style sheet declaration as shown in the following examples:

| | |
|--------------------|--|
| XSLT Syntax | <pre><?xml version="1.0"?> <xsl:stylesheet version="1.0" xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0"> ... </xsl: stylesheet></pre> |
| SLAX Syntax | <pre>version 1.0; ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";</pre> |

You then include variable declarations, a variable call with the `select="jcs:function()"` attribute for XSLT scripts or a simple function call for SLAX scripts, and pass along any required or optional arguments.

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

- `jcs:break-lines()` Function on page 40
- `jcs:close()` Function on page 40

- `jcs:dampen()` Function on page 41
- `jcs:empty()` Function on page 41
- `jcs:execute()` Function on page 42
- `jcs:first-of()` Function on page 42
- `jcs:get-input()` Function on page 43
- `jcs:get-secret()` Function on page 43
- `jcs:hostname()` Function on page 43
- `jcs:invoke()` Function on page 44
- `jcs:open()` Function on page 44
- `jcs:output()` Function on page 45
- `jcs:parse-ip()` Function on page 45
- `jcs:printf()` Function on page 46
- `jcs:progress()` Function on page 46
- `jcs:regex()` Function on page 46
- `jcs:sleep()` Function on page 47
- `jcs:split()` Function on page 47
- `jcs:sysctl()` Function on page 48
- `jcs:syslog()` Function on page 49
- `jcs:trace()` Function on page 50

***jcs:break-lines()* Function**

The `jcs:break-lines()` function breaks a single element into multiple elements, delimited by newlines. This function is especially useful in dealing with large output elements, such as those returned by the `show pfe` command. The function syntax is as follows:

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

where:

- `$lines`—Output broken up into lines.
- `$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 call to the `jcs:open()` function.

***jcs:dampen()* Function**

The `jcs:dampen()` function is used in a script to prevent the same operation from being repeatedly executed. The function returns **true** or **false** based on whether the number of calls to the `jcs:dampen()` function exceeds a *max* number of calls in a time interval *interval*. The function parameters include an arbitrary string, *\$tag*, that is used to distinguish different calls to the `jcs:dampen()` function. This tag is stored in the `/var/run` directory on the router. The function syntax is as follows:

```
var $rc = jcs:dampen($tag, max, interval);
```

where:

- *\$rc*—Return value based on the number of calls to `jcs:dampen()` within a specified time. If the number of calls exceeds *max*, the value is **false**. If the number of calls is less than *max*, the value is **true**.
- *\$tag*—Arbitrary string used to distinguish different calls to the `jcs:dampen()` function.
- *max*—Maximum number of calls of the `jcs:dampen()` function allowed before the function returns **false**. This limit is based on the number of calls within a specified time interval (*interval*).
- *interval*—Time interval, in minutes.

The *max* parameter specifies the maximum number of times that the `jcs:dampen()` function with the corresponding *\$tag* parameter can be called within the time interval *interval* before it returns **false**. In the following example, if the `jcs:dampen()` function with the tag 'mytag1' is called less than three times in a 10-minute interval, the function returns **true**. If the function is called more than three times within 10 minutes, it returns **false**.

```
if (jcs:dampen('mytag1', 3, 10)) {
    /* Code for situations when jcs:dampen() with */
    /* the tag 'mytag1' is called less than three times */
    /* within 10 minutes */
} else {
    /* Code for situations when jcs:dampen() with */
    /* the tag 'mytag1' exceeds the three call maximum */
    /* limit within 10 minutes */
}
```

***jcs:empty()* Function**

The `jcs:empty()` function returns **true** if a node set or string argument is empty. The function syntax is as follows:

```
jcs:empty($set)
```

where *\$set* is the node set 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 the 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` variable is the same as the `$results` variable produced by the `jcs:invoke()` function.
- `$connection`—Connection handle generated by a call to the `jcs:open()` function.
- `$rpc`—RPC to be executed.

jcs:first-of() Function

The `jcs:first-of()` function returns the first nonempty (non-null) item in a list. The function syntax is as follows:

```
var $result = jcs:first-of(object, "expression")
```

where:

- `$results`—First nonempty item in the list.
- `object`—List of objects.
- `expression`—Default value returned if all objects in the list are empty.

In the following example, if the value of `a` is empty, `b` is checked. If the value of `b` is empty, `c` is checked. If the value of `c` is empty, `d` is checked. If the value of `d` is empty, the string "none" is returned.

```
jcs:first-of($a, $b, $c, $d, "none")
```

In the following example, the function returns the description of a logical interface. If a logical interface description does not exist, the function returns the description of the (parent) physical interface. If the parent physical interface description does not exist, the function returns the concatenation of the physical interface name with a period (.) and the logical unit number.

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

***jcs:get-input()* Function**

The `jcs:get-input()` function invokes a CLI prompt and waits for user input. 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 $user-input = jcs:get-input(prompt);
```

where:

- *prompt*—CLI prompt text.
- *\$user-input*—Text typed by the user.

In the following example, the user is prompted to enter a login name. The user's input is stored in the variable `$username`:

```
var $username = jcs:get-input("Enter login id: ");
```

***jcs:get-secret()* Function**

The `jcs:get-secret()` function invokes a CLI prompt and waits for user input. Unlike the `jcs:get-input()` function, the input is not echoed back to the user, which makes the function useful for obtaining a password. 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 $user-input = jcs:get-secret(prompt);
```

where:

- *prompt*—CLI prompt text.
- *\$user-input*—Text typed by the user.

The following example shows how to prompt for a password that is not echoed back to the user:

```
var $password = jcs:get-secret("Enter password: ");
```

***jcs:hostname()* Function**

The `jcs:hostname()` function returns the fully qualified domain name associated with an IPv4 or IPv6 address. The function syntax is as follows:

```
var $name = jcs:hostname($address);
```

where:

- *\$name*—Returned hostname.
- *\$address*—IPv4 or IPv6 address.

***jcs:invoke()* Function**

The `jcs:invoke()` function invokes an RPC. The function 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 including 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 RPCs using the `jcs:execute()` function. The connection handle is closed with the `jcs:close()` function. The function syntax is as follows:

```
var $connection = jcs:open(remote-hostname, username, passphrase)
```

where:

- `$connection`—Connection handle to the remote host.
- `remote-hostname`—Domain name or IP address of the remote router. If you are opening a local connection, do not pass this value.
- `username`—User's login name.
- `passphrase`—User's login password.

The following example shows how to connect to a local device:

```
var $connection = jcs:open()
```

The following example shows how to connect to a remote device:

```
var $connection = jcs:open(remote-hostname)
```

The following example shows how the user `bsmith` with a passphrase `password` obtains a connection handle to the server `fivestar`:

```
var $connection = jcs:open("fivestar", "bsmith", "password")
```

***jcs:output()* Function**

The `jcs:output()` function generates unformatted output text. The text appears in the CLI. The function syntax is as follows:

```
expr jcs:output(string);
```

where *string* is the text that is output to the CLI.

For example:

```
expr jcs:output("The VPN is up.")
```

***jcs:parse-ip()* Function**

The `jcs:parse-ip()` function evaluates an IPv4 or IPv6 address and returns an array containing the following information:

- Host address (or `NULL` in the case of an error)
- Protocol family (`inet` 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*—Array containing the output detailed above.
- *ipaddress*—IPv4 or IPv6 address.
- *prefix-length*—Prefix length.
- *netmask*—Netmask.

In the following example, an IPv4 address and an IPv6 address are parsed, and the resulting output is detailed:

```
var $addr = jcs:parse-ip("10.1.2.10/255.255.255.0");
```

- *\$addr[1]* contains the host address `10.1.2.10`.
- *\$addr[2]* contains the protocol family `inet`.
- *\$addr[3]* contains the prefix length `24`.
- *\$addr[4]* contains the network address `10.1.2.0`.
- *\$addr[5]* contains the netmask for IPv4 `255.255.255.0`.

```
var $addr = jcs:parse-ip("080:0:0:0:8:800:200C:417A/100");
```

- \$addr[1] contains the host address 80::8:800:200C:417A.
- \$addr[2] contains the protocol family inet6.
- \$addr[3] contains the prefix length 100.
- \$addr[4] contains the network address 80::8:800:2000:0.
- \$addr[5] is blank for IPv6 ("").

jcs:printf() Function

The `jcs:printf()` function generates formatted output text. Most standard `printf` formats are supported, in addition to some JUNOS Software-specific formats.

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 containing its single argument to the script log file. The function syntax is as follows:

```
expr jcs:progress(string);
```

where *string* is the text that is output to the script log file.

For example:

```
expr jcs:progress('Working...');
```

jcs:regex() Function

The `jcs:regex()` function evaluates a regular expression against a given string argument and returns any matches. The function syntax is as follows:

```
var $pattern = "(evaluation-pattern)";
var $output = ($pattern, string)
```

where:

- *\$pattern*—Regular expression that is evaluated against the string argument..
- *string*—String within which to search for matches of the specified regular expression.
- *\$output*—Matching values.

For example:

```
var $pattern = "([0-9]+)(:*)([a-z]*)";
var $a = jcs:regex($pattern, "123:xyz");
var $b = jcs:regex($pattern, "r2d2");
var $c = jcs:regex($pattern, "test999!!!");

$a[1] == "123:xyz" # string that matches the full reg expression
$a[2] == "123"    # ([0-9]+)
$a[3] == ":"      # (:*)
$a[4] == "xyz"    # ([a-z]*)
$b[1] == "2d"     # string that matches the full reg expression
$b[2] == "2"      # ([0-9]+)
$b[3] == ""       # (:*) [empty match]
$b[4] == "d"      # ([a-z]*)
$c[1] == "999"    # string that matches the full reg expression
$c[2] == "999"    # ([0-9]+)
$c[3] == ""       # (:*) [empty match]
$c[4] == ""       # ([a-z]*) [empty match]
```

***jcs:sleep()* Function**

The `jcs:sleep()` 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 works over time. To do this, write a script that issues a command, calls the `jcs:sleep()` function, and reissues the same command. The function syntax is as follows:

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

The default time unit is seconds. The *milliseconds* argument is optional. In the following example, `jcs:sleep(1)` causes the script to sleep for 1 second, and `jcs:sleep(0, 10)` causes the script to sleep for 10 milliseconds.

```
<xsl:value-of select="jcs:sleep(1)"/>
<xsl:value-of select="jcs:sleep(0, 10)"/>
```

***jcs:split()* Function**

The `jcs:split()` function splits a string into an array of substrings delimited by the regular expression pattern. The function syntax is as follows:

```
jcs:split(expression, string, <limit>)
```

where:

- *expression*—Regular expression pattern that is used as the delimiter.
- *string*—Original string.
- *limit*—(Optional) Number of substrings into which to break the original string.

If the optional integer argument *limit* is specified, the function splits the entire string into *limit* number of substrings. If there are more than *limit* number of matches, the substrings include the first *limit*-1 matches as well as the remaining portion of the original string for the last match.

In the following example, the original string is "123:abc:456:xyz:789". The `jcs:split()` function breaks this string into substrings that are delimited by the regular expression pattern, which in this case is a colon(`:`). The optional parameter *limit* is not specified, so the function returns an array containing all the substrings that are bounded by the delimiter(`:`).

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789");
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc"
$substrings[3] == "456"
$substrings[4] == "xyz"
$substrings[5] == "789"
```

The following example uses the same original string and regular expression as the previous example, but in this case, the optional parameter *limit* is included. Specifying *limit* = 2 causes the function to return an array containing only two substrings. The substrings include the first match, which is "123" (the same first match as in the previous example) and a second match, which is the remaining portion of the original string after the first occurrence of the delimiter.

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789", 2);
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc:456:xyz:789"
```

***jcs:sysctl()* Function**

The `jcs:sysctl()` function returns a `sysctl` value as a string or an integer. Use the "i" argument to specify an integer. Use the "s" argument to specify a string. The 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.

For example:

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

jcs:syslog() Function

The `jcs:syslog()` function logs messages with a specified priority to the system log file. The function syntax is as follows:

```
jcs:syslog(priority, message, <message2>, <message3>, ... )
```

where:

- **priority**—Priority given to the syslog message. The priority can be specified as a *facility.severity* string, or it can be expressed as an integer calculated from the corresponding numeric values of the facility and severity strings. Table 6 on page 49 and Table 7 on page 50 show the facility and severity strings available and their corresponding numeric values.

The integer value of the *priority* parameter is calculated by multiplying the facility string numeric value by 8 and adding the severity string numeric value. For example, if the *facility.severity* string pair is "pfe.alert", the priority value is 161 ((20 x 8) + 1).

- **message**—String or variable that is output to the system log file.
- **message2**—(Optional) Any additional number of strings or variable names passed as arguments to the function. These are concatenated with the **message** argument and output to the syslog.

The *message* argument is a string or variable that is written to the system log file. Optionally, additional strings or variables can be included in the argument list. The *message* argument is concatenated with any additional parameters, and the concatenated string is written to the system log file. The syslog is specified at the [edit system syslog] hierarchy level of the router configuration file.

The following three examples log pfe messages with an alert priority. The string "mymessage" is output to the system log file. All three examples are equivalent:

```
expr jcs:syslog("pfe.alert", "mymessage");
```

```
expr jcs:syslog(161, "mymessage");
```

```
var $message = "mymessage";
expr jcs:syslog("pfe.alert", $message);
```

The following example logs pfe messages with an alert priority as in the previous example. In this example, however, there are additional string parameters. For this case, the concatenated string "mymessage mymessage2" is output to the system log file.

```
expr jcs:syslog("pfe.alert", "mymessage ", "mymessage2");
```

Table 6: Facility Strings

| Facility String | Description | Numeric Value |
|-----------------|----------------------|---------------|
| auth | Authorization system | 4 |

Table 6: Facility Strings (*continued*)

| Facility String | Description | Numeric Value |
|-----------------|-----------------------------|---------------|
| 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 the UI | 23 |
| pfe | Packet Forwarding Engine | 20 |
| user | User processes | 1 |

Table 7: Severity Strings

| Severity String | Description | Numeric Value |
|-----------------|---|---------------|
| alert | Conditions that should be corrected immediately | 1 |
| crit | Critical conditions | 2 |
| debug | Debug messages | 7 |
| emerg or panic | Panic conditions | 0 |
| err or error | Error conditions | 3 |
| info | Informational messages | 6 |
| notice | Conditions that should be specially handled | 5 |
| warn or warning | Warning messages | 4 |

***jcs:trace()* Function**

The `jcs:trace()` function issues a trace message, which is sent to the trace file.

For example:

```
<xsl:value-of select="jcs:trace('test')"/>
```

Importing the junos.xsl File

The import file `junos.xsl` contains several useful templates that you can call within commit scripts, op scripts, and event scripts. 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 in the following examples:

| | |
|--------------------|--|
| XSLT Syntax | <pre><?xml version="1.0"?> <xsl:stylesheet version="1.0" xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0"> <xsl:import href="../import/junos.xsl"/> ... </xsl:stylesheet></pre> |
| SLAX Syntax | <pre>version 1.0; ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0"; import "../import/junos.xsl";</pre> |

Templates in the junos.xsl File

Named templates in the `junos.xsl` file use the `jcs:` prefix to avoid conflicting with user-defined templates of the same name in a script. The templates in the `jcs:` namespace allow you to accomplish scripting tasks more easily. To use these templates in your scripts, you 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 in the `junos.xsl` file that allows you to use shortened XPath expressions in your scripts.

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

- `<jcs:edit-path>` Template on page 51
- `<jcs:emit-change>` Template on page 52
- `<jcs:emit-comment>` Template on page 54
- `<jcs:statement>` Template on page 55
- `<xsl:template match="/">` Template on page 55

`<jcs:edit-path>` Template

The `<jcs:edit-path>` 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:

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/mps)">
      <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>
            <mps/>
          </family>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

When you commit a configuration that includes one or more interfaces that have IS-IS enabled but do not have the `family mpls` statement included at the `[edit interfaces interface-name unit logical-unit-number]` hierarchy level, the `<jcs:emit-change>` template adds the `family mpls` statement to the configuration and generates the following CLI output:

```
[edit]
user@host# commit
[edit interfaces interface so-1/2/3 unit 0]
  warning: Adding 'family mpls' to ISO-enabled interface
[edit interfaces interface so-1/2/3 unit 0]
  warning: Adding ISO-enabled interface so-1/2/3.0 to [protocols mpls]
[edit interfaces interface so-1/3/2 unit 0]
  warning: Adding 'family mpls' to ISO-enabled interface
[edit interfaces interface so-1/3/2 unit 0]
  warning: Adding ISO-enabled interface so-1/3/2.0 to [protocols mpls]
commit complete
```

The `content` parameter of the `<jcs:emit-change>` template provides a simpler method for specifying a change to the configuration. For example, consider the following code:

```
<xsl:with-param name="content">
  <family>
    <mps/>
  </family>
</xsl:with-param>
```

The `<jcs:emit-change>` template converts the `content` parameter into a `<change>` request. The `<change>` request inserts the provided partial configuration content into the complete hierarchy of the current context node. Thus, the `<jcs:emit-change>` template changes the hierarchy information in the `content` parameter into the following code:

```
<change>
  <interfaces>
    <interface>
      <name><xsl:value-of select="name"/></name>
      <unit>
        <name><xsl:value-of select="unit/name"/></name>
        <family>
          <mpls/>
        </family>
      </unit>
    </interface>
  </interfaces>
</change>
```

If a transient change is required, the `tag` parameter can be passed in as `'transient-change'`, as shown here:

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

The extra quotation marks are required to allow XSLT to distinguish between the string `"transient-change"` and the contents of a node named `"transient-change"`. If the change is relative to a node other than the context node, the parameter `"dot"` can be set to that node, as shown in the following example, where context is set to the `[edit chassis]` hierarchy level:

```
<xsl:for-each select="interfaces/interface/unit">
  ...
  <xsl:call-template name="jcs:emit-change">
    <xsl:with-param name="dot" select="chassis"/>
  ...
</xsl:for-each>
```

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



```

    </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

```

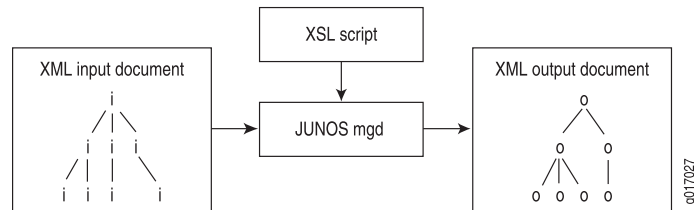
The test of the MTU setting is not performed in the `<xnm:error>` element. For the full example, see “Example: Imposing a Minimum MTU Setting” on page 189.

<xsl:template match="/"> Template

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

The JUNOS management process (mgd) generates the output document as the product of its evaluation of the input document, as shown in Figure 3 on page 56.

Figure 3: Commit Script Input and Output



Generally, an XSLT engine uses recursion to evaluate the entire input document. However, the `<xsl:apply-templates>` instruction allows you to limit the scope of the evaluation so that the management process (the JUNOS 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-results>
3      <xsl:apply-templates select="commit-script-input/configuration"/>
4    </commit-script-results>
5  </xsl:template>
  
```

Line 1 matches the root node of the input document. When the management process sees the root node of the input document, this template is applied.

```
1  <xsl:template match="/">
```

Line 2 designates the root, top-level tag of the output document. Thus, Line 2 specifies that the evaluation of the input document results in an output document whose top-level tag is `<commit-script-results>`.

```
2    <commit-script-results>
```

Line 3 limits the scope of the evaluation of the input document to the contents of the `<configuration>` element, which is a child of the `<commit-script-input>` element.

```
3      <xsl:apply-templates select="commit-script-input/configuration"/>
```

Lines 4 and 5 are closing tags.

You do not need to explicitly include the `<xsl:template match="/">` template in your scripts because this template is included in the import file `junos.xml`.

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.xml` 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-results>  <!-- from template in junos.xml 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-results>  <!-- end of template in junos.xml import file - ->
```

When you import the `junos.xml` file and explicitly include the `<xsl:template match="configuration">` tag in your commit script, the context (`dot`) moves to the `<configuration>` node. This allows you to write all XPath expressions relative to that point. This technique allows you to simplify the XPath expressions you use in your commit scripts. For example, instead of writing this, which matches the 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')">
```


Chapter 7

Summary of XPath and XSLT Constructs

This chapter discusses the following topics:

- Summary of Standard XPath and XSLT Functions Referenced in This Guide on page 60
- Summary of Standard XSLT Elements and Attributes Referenced in This Guide on page 64
- Summary of JUNOS XSLT Extension Functions on page 76
- Summary of JUNOS Named XSLT Templates on page 87

Summary of Standard XPath and XSLT Functions Referenced in This Guide

JUNOS commit scripts, op scripts, and event scripts support all functions defined in the Extensible Markup Language Path Language (XPath) and Extensible Stylesheet Language Transformations (XSLT) scripting languages. This section describes only the functions referenced in this guide. The functions are organized alphabetically.

concat()

| | |
|-----------------------|--|
| Syntax | <i>string</i> concat (<i>string</i> , <i>string</i> +) |
| Description | Return the concatenation of the arguments. |
| Usage Examples | See “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Controlling IS-IS and MPLS Interfaces” on page 203, “Example: Adding T1 Interfaces to a RIP Group” on page 206, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Configuring Dual Routing Engines” on page 234. |

contains()

| | |
|-----------------------|--|
| Syntax | <i>boolean</i> contains (<i>string</i> , <i>string</i>) |
| Description | Return TRUE if the first argument string contains the second argument string, and otherwise returns FALSE. |
| Usage Examples | See “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ starts-with() on page 62 |

count()

| | |
|-----------------------|--|
| Syntax | <i>number</i> count (<i>node-set</i>) |
| Description | Return the number of nodes in the argument node-set. |
| Usage Examples | See “Example: Limiting the Number of E1 Interfaces” on page 191. |
| Related Topics | ■ last() on page 61 ■ position() on page 62 |

last()

| | |
|-----------------------|---|
| Syntax | <i>number last()</i> |
| Description | Return the index of the last node in the list that is currently being evaluated. |
| Usage Examples | See “Example: Limiting the Number of E1 Interfaces” on page 191. |
| Related Topics | <ul style="list-style-type: none"> ■ count() on page 60 ■ position() on page 62 |

name()

| | |
|-----------------------|--|
| Syntax | <i>string name(<node-set>)</i> |
| 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 52. |

not()

| | |
|-----------------------|---|
| Syntax | <i>boolean not(boolean)</i> |
| Description | Return TRUE if its argument is FALSE, and FALSE if the argument is TRUE. |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Controlling IS-IS and MPLS Interfaces” on page 203, “Example: Configuring a Default Encapsulation Type” on page 209, “Example: Controlling LDP Configuration” on page 212, “Example: Adding a Final then accept Term to a Firewall” on page 216, “Example: Configuring Administrative Groups for LSPs” on page 230, “Example: Configuring Dual Routing Engines” on page 234, and “Example: Preventing Import of the Full Routing Table” on page 238. |

position()

| | |
|-----------------------|---|
| Syntax | <i>number position()</i> |
| Description | Return the position of the context node among the list of nodes that are currently being evaluated. |
| Usage Examples | See “Example: Adding a Final then accept Term to a Firewall” on page 216 and “Example: Prepending a Global Policy” on page 247. |
| Related Topics | <ul style="list-style-type: none"> ■ <code>count()</code> on page 60 ■ <code>last()</code> on page 61 |

starts-with()

| | |
|-----------------------|---|
| Syntax | <i>boolean starts-with(string, string)</i> |
| Description | Return TRUE if the first argument string starts with the second argument string, and returns FALSE otherwise. |
| Usage Examples | See “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Limiting the Number of ATM Virtual Circuits” on page 200, “Example: Adding T1 Interfaces to a RIP Group” on page 206, “Example: Configuring a Default Encapsulation Type” on page 209, and “Example: Configuring Dual Routing Engines” on page 234. |
| Related Topics | <ul style="list-style-type: none"> ■ <code>contains()</code> on page 60 |

string-length()

| | |
|-----------------------|---|
| Syntax | <i>number string-length(<string>)</i> |
| Description | Return the number of characters in the string. If the argument is omitted, it returns the string value of the context node. |
| Usage Examples | See “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |

substring-after()

| | |
|-----------------------|---|
| Syntax | <i>string</i> substring-after(<i>string</i> , <i>string</i>) |
| 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 “Example: Limiting the Number of E1 Interfaces” on page 191 and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ substring-before() on page 63 |

substring-before()

| | |
|-----------------------|--|
| Syntax | <i>string</i> substring-before(<i>string</i> , <i>string</i>) |
| 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 “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ substring-after() on page 63 |

Summary of Standard XSLT Elements and Attributes Referenced in This Guide

JUNOS commit scripts, op scripts, and event scripts support all elements and attributes defined in the Extensible Markup Language Path Language (XPath) and Extensible Stylesheet Language Transformations (XSLT) scripting languages. This section describes only the elements and attributes referenced in this guide. The elements and attributes are organized alphabetically.

<xsl:apply-templates>

| | |
|-----------------------|--|
| Syntax | <pre> <xsl:apply-templates select="<i>node-set-expression</i>"> <xsl:with-param name="<i>qualified-name</i>" select="<i>expression</i>"> ... </xsl:with-param> </xsl:apply-templates> </pre> |
| 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 “Example: Adding a Final then accept Term to a Firewall” on page 216 and “Example: Preventing Import of the Full Routing Table” on page 238. |
| Related Topics | <ul style="list-style-type: none"> ■ <xsl:call-template> ■ <xsl:for-each> ■ <xsl:template> ■ <xsl:with-param> |

<xsl:call-template>

| | |
|-----------------------|---|
| Syntax | <pre> <xsl:call-template name="<i>qualified-name</i>"> <xsl:with-param name="<i>qualified-name</i>" select="<i>expression</i>"> ... </xsl:with-param> </xsl:call-template> </pre> |
| 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 “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | <ul style="list-style-type: none"> ■ <xsl:apply-templates> ■ <xsl:template> |

<xsl:choose>

| | |
|-----------------------|--|
| Syntax | <pre> <xsl:choose> <xsl:when test="<i>boolean-expression</i>"> ... </xsl:when> <xsl:otherwise> ... </xsl:otherwise> </xsl:choose> </pre> |
| 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 “Example: Configuring Dual Routing Engines” on page 234, “Example: Preventing Import of the Full Routing Table” on page 238, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | <ul style="list-style-type: none"> ■ <xsl:if> ■ <xsl:otherwise> ■ <xsl:when> |

<xsl:comment>

| | |
|-----------------------|--|
| Syntax | <pre><xsl:comment> ... </xsl:comment></pre> |
| Description | <p>Generate a comment within the final document. The content within the <code><xsl:comment></code> 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.</p> <p>XSLT files can contain ordinary <code><!-- ... Insert your comment here ... --></code> comments, but these are ignored by the processor. To generate a comment within the final document, use an <code><xsl:comment></code> element.</p> |
| Usage Examples | See “Example: Adding a Final then accept Term to a Firewall” on page 216. |

<xsl:copy-of>

| | |
|-----------------------|--|
| Syntax | <code><xsl:copy-of select="expression"/></code> |
| Description | Create a copy of what is selected by the expression defined in the <code>select</code> attribute. Namespace nodes, child nodes, and attributes of the current node are automatically copied as well. |
| Attributes | <code>select</code> —Specifies an expression to select nodes to be copied. |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183. |
| Related Topics | ■ <code><xsl:value-of></code> |

<xsl:element>

| | |
|-----------------------|---|
| Syntax | <code><xsl:element name="expression"/></code> |
| Description | Create an element node in the output document. |
| Attributes | <code>name</code> —Specifies the name of the element to be created. The value of the <code>name</code> 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 <code><xsl:element name="{ \$isis-level-1 }"</code> . |
| Usage Examples | See “Example: Creating a Complex Configuration Based on a Simple Interface Configuration” on page 224. |

<xsl:for-each>

| | |
|-----------------------|--|
| Syntax | <pre><xsl:for-each select="node-set-expression"> ... </xsl:for-each></pre> |
| 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 “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Adding T1 Interfaces to a RIP Group” on page 206, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Configuring Dual Routing Engines” on page 234. |
| Related Topics | ■ <xsl:apply-templates> |

<xsl:if>

| | |
|-----------------------|--|
| Syntax | <pre><xsl:if test="boolean-expression"> ... </xsl:if></pre> |
| 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 “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Adding T1 Interfaces to a RIP Group” on page 206, and “Example: Configuring Dual Routing Engines” on page 234. |
| Related Topics | ■ <xsl:choose> ■ <xsl:when> |

<xsl:import>

| | |
|-----------------------|---|
| Syntax | <code><xsl:import href="../../import/junos.xml"/></code> |
| Description | <p>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 <code><xsl:import></code> elements must be the first elements within the style sheet, the first children of the <code><xsl:stylesheet></code> document element. The path can be any URI. The <code>../../import/junos.xml</code> path shown in the syntax is standard for all commit scripts, op scripts, and event scripts.</p> <p>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.</p> |
| Attributes | href—Specifies the location of the imported style sheet. |
| Usage Examples | See all examples listed in “Commit Script Examples” on page 183. |
| Related Topics | <ul style="list-style-type: none"> ■ <code><xsl:stylesheet></code> |

<xsl:otherwise>

| | |
|-----------------------|---|
| Syntax | <pre> <xsl:otherwise> ... </xsl:otherwise> </pre> |
| Description | Within an <code><xsl:choose></code> instruction, include the instructions that are processed if none of the expressions defined in the <code>test</code> attributes of the <code><xsl:when></code> elements evaluate as TRUE. |
| Usage Examples | See “Example: Configuring Dual Routing Engines” on page 234 and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | <ul style="list-style-type: none"> ■ <code><xsl:choose></code> ■ <code><xsl:when></code> |

<xsl:param>

| | |
|-----------------------|---|
| Syntax | <pre> <xsl:param name="<i>qualified-name</i>" select="<i>expression</i>"> ... </xsl:param> </pre> |
| 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 | <p>name—Defines the name of the parameter.</p> <p>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.</p> |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Limiting the Number of ATM Virtual Circuits” on page 200, and “Example: Preventing Import of the Full Routing Table” on page 238. |
| Related Topics | <ul style="list-style-type: none"> ■ <xsl:template> ■ <xsl:variable> ■ <xsl:with-param> |

<xsl:stylesheet>

Syntax

```
<xsl:stylesheet version="1.0">
  <xsl:import href="../import/junos.xml"/>
  ...
</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.xml** 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 183.

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:template>
```

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 183.

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 “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Controlling IS-IS and MPLS Interfaces” on page 203, and “Example: Adding a Final then accept Term to a Firewall” on page 216.

<xsl:value-of>

| | |
|-----------------------|---|
| Syntax | <code><xsl:value-of select="<i>string-expression</i>" /></code> |
| 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 <code><xsl:value-of></code> 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 <code><xsl:copy-of></code> instruction instead. |
| Attributes | select —Specifies the expression that is evaluated. |
| Usage Examples | See “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Controlling IS-IS and MPLS Interfaces” on page 203, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ <code><xsl:copy-of></code> |

<xsl:variable>

| | |
|-----------------------|---|
| Syntax | <pre> <xsl:variable name="<i>qualified-name</i>" select="<i>expression</i>"> ... </xsl:variable> </pre> |
| Description | Declare a local or global variable. If the <code><xsl:variable></code> instruction appears at the top level of the style sheet as a child of the <code><xsl:stylesheet></code> 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 | <p>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.</p> <p>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 <code><xsl:variable></code> 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.</p> |
| Usage Examples | See “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Limiting the Number of ATM Virtual Circuits” on page 200, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ <code><xsl:param></code> |

<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 “Example: Configuring Dual Routing Engines” on page 234, “Example: Preventing Import of the Full Routing Table” on page 238, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241.

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 “Example: Configuring Dual Routing Engines” on page 234, “Example: Preventing Import of the Full Routing Table” on page 238, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241.

Related Topics ■ <xsl:apply-templates>
 ■ <xsl:call-template>
 ■ <xsl:param>

Summary of JUNOS XSLT Extension Functions

The JUNOS extension functions provided for use in commit, op, and event scripts enable you to perform operations that are difficult or impossible to achieve with XPath. All JUNOS extension functions have the **jcs:** prefix on their names to indicate they belong to the **jcs:** namespace. This section lists the functions in alphabetical order.

jcs:break-lines()

| | |
|-----------------------|--|
| Syntax | <code>jcs:break-lines(<i>expression</i>)</code> |
| Description | Break a simple element into multiple elements, delimited by newlines. This is especially useful for large output elements such as those returned by the <code>show pfe</code> command. |
| Usage Examples | <pre>var \$lines = jcs:break-lines(\$output); for-each (\$lines) { ... }</pre> |

jcs:close()

| | |
|-----------------------|---|
| Syntax | <code>jcs:close(<i>connection</i>)</code> |
| Description | Close a previously opened connection handle. |
| Usage Examples | <p>In the following example, <code>\$connection</code> is a connection handle generated by a call to the <code>jcs:open()</code> function:</p> <pre>expr jcs:close(\$connection);</pre> |

jcs:dampen()

Syntax `jcs:dampen($tag, max, interval)`

Description Prevent the same operation from being repeatedly executed. The `dampen` function returns `true` or `false` based on whether the number of calls to the `jcs:dampen()` function exceeds a *max* number of calls in the time interval *interval*. The function parameters include an arbitrary string, *\$tag*, that is used to distinguish different calls to the `jcs:dampen()` function. This tag is stored in the `/var/run` directory on the router. The *max* parameter specifies the maximum number of times that the `jcs:dampen()` function with the corresponding *\$tag* parameter can be called within the time interval *interval* before it returns `false`. The time interval is specified in minutes.

Usage Examples In the following example, if the `jcs:dampen()` function with the tag 'mytag1' is called less than three times in a 10-minute interval, the function returns `true`. If the function is called more than three times within 10 minutes, it returns `false`.

```
if (jcs:dampen('mytag1', 3, 10)) {
    /* Code for situations when jcs:dampen() with */
    /* the tag 'mytag1' is called less than three times */
    /* within 10 minutes */
} else {
    /* Code for situations when jcs:dampen() with */
    /* the tag 'mytag1' exceeds the three call maximum */
    /* limit within 10 minutes */
}
```

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($set) ) {
    /* Code to handle true value ($set is empty) */
}
```

jcs:execute()

| | |
|-----------------------|--|
| Syntax | <code>jcs:execute(connection, rpc)</code> |
| Description | Execute an RPC within the context of a specified connection handle. Any number of RPCs may be executed within the context of the connection handle until it is closed. |
| Usage Examples | <code>var \$results = jcs:execute(\$connection, \$rpc);</code> |

jcs:first-of()

| | |
|-----------------------|--|
| Syntax | <code>jcs:first-of(object, + "expression")</code> |
| Description | Return the first nonempty (non-null) item in a list. |
| Usage Examples | <p>In the following example, if the value of <code>a</code> is empty, <code>b</code> is checked. If the value of <code>b</code> is empty, <code>c</code> is checked. If the value of <code>c</code> is empty, <code>d</code> is checked. If the value of <code>d</code> is empty, the string "none" is returned.</p> <pre>jcs:first-of(\$a, \$b, \$c, \$d, "none")</pre> <p>In the following example, the function returns the description of a logical interface. If a logical interface description does not exist, the function returns the description of the (parent) physical interface. If the parent physical interface description does not exist, the function returns the concatenation of the physical interface name with a period (.) and the logical unit number.</p> <pre><xsl:variable name="description" select="jcs:first-of(description, ../description, concat(..name, '.', name))"/></pre> <p>See also “Example: Displaying DNS Hostname Information in an Op Script” on page 313.</p> |

jcs:get-input()

| | |
|-----------------------|--|
| Syntax | <code>jcs:get-input(string)</code> |
| Description | Invoke a CLI prompt and wait for user input. The user input is defined as a string for subsequent use. |
| Usage Examples | <code>var \$user-input = jcs:get-input("Enter input: ");</code> |

jcs:get-secret()

| | |
|-----------------------|---|
| Syntax | <code>jcs:get-secret(<i>string</i>)</code> |
| Description | Invoke a CLI prompt and wait for user input. The input is not echoed back to the user, which makes the function useful for obtaining passwords. The user input is defined as a string for subsequent use. |
| Usage Examples | <code>var \$password = jcs:get-secret("Enter password: ");</code> |

jcs:hostname()

| | |
|-----------------------|--|
| Syntax | <code>jcs:hostname(<i>expression</i>)</code> |
| Description | Return the fully qualified domain name associated with a given IPv4 or IPv6 address. |
| Usage Examples | <pre><xsl:variable name="name" select="jcs:hostname(\$dest)"/> <xsl:value-of select="concat(\$address, ' is ', jcs:hostname(\$address))"/></pre> |

See also “Example: Finding LSPs to Multiple Destinations in an Op Script” on page 326.

jcs:invoke()

Syntax `jcs:invoke(rpc)`

Description Invoke a remote procedure call (RPC). The function 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 including 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')"/>
```

See also “Example: Customizing Output of the `show interfaces terse` Command in an Op Script” on page 316.

jcs:open()

Syntax `jcs:open(remote-hostname, username, passphrase)`

Description Return a connection handle that can be used to execute RPCs using the `jcs:execute()` extension function. The connection handle is closed with the `jcs:close()` function.

Usage Examples The following example shows how the user **bsmith** with a passphrase **password** obtains a connection handle to the server **fivestar**.

```
var $connection = jcs:open("fivestar", "bsmith", "password");
```

jcs:output()

Syntax `jcs:output('expression')`

Description Generate unformatted output text. The text appears in the CLI.

Usage Examples XSLT syntax:

```
<xsl:value-of select="jcs:output('The VPN is up.')" />
```

SLAX syntax:

```
expr jcs:output('The VPN is up.');
```

jcs:parse-ip()

Syntax `jcs:parse-ip("ipaddress/(prefix-length | netmask)")`

Description Evaluate an IPv4 or IPv6 address and return an array containing:

- Host address (or NULL in the case of an error)
- Protocol family (inet for IPv4 or inet6 for IPv6)
- Prefix length
- Network address
- Netmask (for IPv4 address; left blank for IPv6 addresses)

Usage Examples In the following example, an IPv4 address and an IPv6 address are parsed and the resulting output is detailed:

```
var $addr = jcs:parse-ip("10.1.2.10/255.255.255.0");
```

- \$addr[1] contains the host address 10.1.2.10.
- \$addr[2] contains the protocol family inet.
- \$addr[3] contains the prefix length 24.
- \$addr[4] contains the network address 10.1.2.0.
- \$addr[5] contains the netmask for IPv4 255.255.255.0.

```
var $addr = jcs:parse-ip("080:0:0:0:8:800:200C:417A/100");
```

- \$addr[1] contains the host address 80::8:800:200C:417A.
- \$addr[2] contains the protocol family inet6.
- \$addr[3] contains the prefix length 100.
- \$addr[4] contains the network address 80::8:800:2000:0.
- \$addr[5] is blank for IPv6 ("").

jcs:printf()

| | |
|-----------------------|---|
| Syntax | <code>jcs:printf('expression')</code> |
| Description | <p>Generate formatted output text. Most standard <code>printf</code> formats are supported, in addition to some JUNOS Software-specific formats.</p> <p>The <code>%j1</code> operator emits the field only if the field was changed from the last time the function was run.</p> <p>The <code>%jc</code> operator capitalizes the first letter of the format output.</p> <p>The <code>%jt{TAG}</code> operator emits the tag if the field is not empty.</p> |
| Usage Examples | <pre><xsl:value-of select="jcs:printf('%-24j1s %-5jcs %-5jcs %s%jt{ -> }s\n', 'so-0/0/0', 'up', 'down', '10.1.2.3', '')"/></pre> |

jcs:progress()

| | |
|-----------------------|--|
| Syntax | <code>jcs:progress('expression')</code> |
| Description | Issue a progress message containing the single argument to the script log file. |
| Usage Examples | <p>XSLT syntax:</p> <pre><xsl:value-of select="jcs:progress('Working...')"/></pre> <p>SLAX syntax:</p> <pre>expr jcs:progress('Working...');</pre> |

jcs:regex()

Syntax `jcs:regex(expression, string)`

Description Return the set of strings within *string* that are matched by the given regular expression. This function requires two arguments: the regular expression and the string within which to search for the expression.

Usage Examples

```

var $pattern = "([0-9]+)(:*)([a-z]*)";
var $a = jcs:regex($pattern, "123:xyz");
var $b = jcs:regex($pattern, "r2d2");
var $c = jcs:regex($pattern, "test999!!!");

$a[1] == "123:xyz"   # string that matches the full reg expression
$a[2] == "123"      # ([0-9]+)
$a[3] == ":"        # (:*)
$a[4] == "xyz"      # ([a-z]*)
$b[1] == "2d"       # string that matches the full reg expression
$b[2] == "2"        # ([0-9]+)
$b[3] == ""         # (:*) [empty match]
$b[4] == "d"        # ([a-z]*)
$c[1] == "999"      # string that matches the full reg expression
$c[2] == "999"      # ([0-9]+)
$c[3] == ""         # (:*) [empty match]
$c[4] == ""         # ([a-z]*) [empty match]
```

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 works 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)` causes the script to sleep for 1 second, and `jcs:sleep(0, 10)` causes the script to sleep for 10 milliseconds.

```

<xsl:value-of select="jcs:sleep(1)"/>
<xsl:value-of select="jcs:sleep(0, 10)"/>
```

jcs:split()

Syntax `jcs:split(expression, string, <limit>)`

Description Split a string into an array of substrings delimited by a regular expression pattern. If the optional integer argument *limit* is specified, the function splits the entire string into *limit* number of substrings. If there are more than *limit* number of matches, the substrings include the first *limit*-1 matches as well as the remaining portion of the original string for the last match.

Usage Examples In the following example, the original string is "123:abc:456:xyz:789". The `jcs:split()` function breaks this string into substrings that are delimited by the regular expression pattern, which in this case is a colon(:). The optional parameter *limit* is not specified, so the function returns an array containing all the substrings that are bounded by the delimiter(:).

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789");
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc"
$substrings[3] == "456"
$substrings[4] == "xyz"
$substrings[5] == "789"
```

The following example uses the same original string and regular expression as the previous example, but in this case, the optional parameter *limit* is included. Specifying *limit* = 2 causes the function to return an array containing only two substrings. The substrings include the first match, which is "123" (the same first match as in the previous example) and a second match, which is the remaining portion of the original string after the first occurrence of the delimiter.

```
var $pattern = "(:)";
var $substrings = jcs:split($pattern, "123:abc:456:xyz:789", 2);
```

returns:

```
$substrings[1] == "123"
$substrings[2] == "abc:456:xyz:789"
```

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:syslog()

Syntax `jcs:syslog(priority, message, <message2>, <message3> ...)`

Description Log messages with the specified priority to the system log file. The priority can be expressed as a *facility.severity* string or as a calculated integer (see “jcs:syslog() Function” on page 49 for more information about calculating the priority as an integer). The *message* argument is a string or variable that is written to the system log file. Optionally, additional strings or variables can be included in the argument list. The *message* argument is concatenated with any additional parameters, and the concatenated string is written to the system log file. The syslog is specified at the [edit system syslog] hierarchy level of the router configuration file.

Usage Examples The following three examples log pfe messages with an alert priority. The string "mymessage" is output to the system log file. All three examples are equivalent:

```
expr jcs:syslog("pfe.alert", "mymessage");
expr jcs:syslog(161, "mymessage");
var $message = "mymessage";
expr jcs:syslog("pfe.alert", $message);
```

The following example logs pfe messages with an alert priority similar to the previous example. In this example, however, there are additional string parameters. For this case, the concatenated string "mymessage mymessage2" is output to the system log file.

```
expr jcs:syslog("pfe.alert", "mymessage ", "mymessage2");
```

jcs:trace()

Syntax `jcs:trace('expression')`

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

Usage Examples `<xsl:value-of select="jcs:trace('test')"/>`

Summary of JUNOS Named XSLT Templates

Named XSLT templates are provided in the `junos.xsl` import file to make scripting tasks easier in commit, op, and event scripts. All named templates have the `jcs:` prefix on their names to indicate they belong to the `jcs:` namespace. This section lists the templates in alphabetical order.

When you use named templates in a script, you must include the `xmlns:jcs` attribute in the opening `<xsl:stylesheet>` tag element and the `<xsl:import/>` tag element to import the `junos.xsl` file, as in this example:

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

You then include an `<xsl:call-template name="name">` element in the script to reference each named template, passing in any required or optional parameters.

For more information about attributes and tag elements to include in your scripts, see “Required Boilerplate for Commit Scripts” on page 115, “Required Boilerplate for Op Scripts” on page 291, and “Required Boilerplate for Event Scripts” on page 415.

`<jcs:edit-path>`

| | |
|-------------------------|---|
| Syntax | <pre><xsl:call-template name="jcs:edit-path"> <xsl:with-param name="dot" select="expression"/> </xsl:call-template></pre> |
| Description | Generate an <code><edit-path></code> element suitable for inclusion in an <code><xnm:error></code> or <code><xnm:warning></code> element. By default, the location of the configuration error is passed as <code>dot</code> into the <code><jcs:edit-path></code> template. This location defaults to <code>“.”</code> , the current position in the XML hierarchy. You can alter the default by including the <code>select</code> attribute of the <code>dot</code> parameter. |
| Parameters | <code><xsl:param name="dot" select="."></code> —Allows you to indicate a location other than the current location in the XML hierarchy. The <code>select</code> attribute contains the current context <code>“.”</code> as a default value. If you want to change the current context, you can include the <code>dot</code> parameter and include a different XPath expression in the <code>select</code> attribute. |
| Usage Guidelines | See “ <code><jcs:edit-path></code> Template” on page 51. |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183. |
| Related Topics | <ul style="list-style-type: none"> ■ <code><xsl:param></code> ■ <code><xsl:with-param></code> |

<jcs:emit-change>

| | |
|-------------------------|--|
| Syntax | <pre> <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> </pre> |
| Description | Generate a <change> or <transient-change> element, which results in a persistent or transient change to the configuration. |
| Parameters | <p><xsl:param name="content">—Allows you to include the content of the change, relative to dot.</p> <p><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.</p> <p><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.</p> <p><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).</p> <p><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:</p> <pre> <xsl:with-param name="tag" select="'transient-change'"/> </pre> |
| Usage Guidelines | See “<jcs:emit-change> Template” on page 52. |
| Usage Examples | See “Example: Imposing a Minimum MTU Setting” on page 189. |
| 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 54.

Usage Examples See “Example: Adding a Final then accept Term to a Firewall” on page 216.

Related Topics ■ `<jcs:emit-change>` on page 88

`<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 55.

Usage Examples See “Example: Configuring Dual Routing Engines” on page 234.

Related Topics ■ `<xsl:param>`
 ■ `<xsl:with-param>`

Chapter 8

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 | <code>apply-templates <i>expression</i>;</code> |
| 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 | <i>expression</i> —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 | <pre>match configuration { apply-templates system/host-name; }</pre> |
| XSLT Equivalent | <pre><xsl:template match="configuration"> <xsl:apply-templates select="system/host-name"/> </xsl:template></pre> |
| Usage Examples | See “Example: Adding a Final then accept Term to a Firewall” on page 216 and “Example: Preventing Import of the Full Routing Table” on page 238. |
| Related Topics | <ul style="list-style-type: none">■ match on page 97■ mode on page 98■ with on page 104 |

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 “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241.

- Related Topics**
- `template` on page 101
 - `with` on page 104

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 “Example: Configuring Dual Routing Engines” on page 234 and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241.

Related Topics ■ [if](#) on page 96

for-each

| | |
|------------------------|--|
| Syntax | <pre>for-each (expression) { /* code */ }</pre> |
| Description | Include a looping mechanism that repeats script processing for each instance of identical XML elements. The element nodes are selected by the <i>expression</i> attribute. Each of the nodes is then processed by the instructions contained in the for-each statement. |
| Attributes | <i>expression</i> —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 | <pre>for-each (\$inventory/chassis/chassis-module /chassis-sub-module[part-number == '750-000610']) { <message> "Down rev PIC in " _ ../name _ ", " _ name _ ": " _ description; }</pre> |
| XSLT Equivalent | <pre><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></pre> |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Adding T1 Interfaces to a RIP Group” on page 206, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Configuring Dual Routing Engines” on page 234. |

if

| | |
|------------------------|---|
| Syntax | <pre> if (expression) { /* code */ } </pre> |
| Description | Include a conditional construct that causes instructions to be processed if the Boolean expression held in the test attribute evaluates to TRUE. |
| Attributes | <i>expression</i> —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 | <pre> 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 */ } } </pre> |
| XSLT Equivalent | <pre> <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> </pre> |
| Usage Examples | See “Example: Configuring Dual Routing Engines” on page 234, “Example: Preventing Import of the Full Routing Table” on page 238, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ else on page 94 |

match

| | |
|------------------------|--|
| Syntax | <pre>match <i>expression</i> { <i>statements</i>; }</pre> |
| Description | Declare a template that contains rules to apply when a specified node is matched. The <code>match</code> statement associates the template with an XML element. The <code>match</code> statement can also be used to define a template for a whole branch of the XML document. For example, <code>match /</code> matches the whole document. |
| Attributes | <i>expression</i> —Specifies a pattern against which nodes are matched. |
| SLAX Example | <pre>match host-name { <hello> .; }</pre> |
| XSLT Equivalent | <pre><xsl:template match="host-name"> <hello> <xsl:value-of select="."/> </hello> </xsl:template></pre> |
| Usage Examples | See all examples listed in “Commit Script Examples” on page 183. |
| Related Topics | <ul style="list-style-type: none"> ■ apply-templates on page 91 ■ mode on page 98 |

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 “Example: Adding a Final then accept Term to a Firewall” on page 216.

- Related Topics**
- [apply-templates on page 91](#)
 - [match on page 97](#)

param

| | |
|------------------------|---|
| Syntax | <code>param \$name=value;</code> |
| Description | <p>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.</p> <p>In SLAX, parameter and variable names contain the dollar sign (\$) even in the declaration. This is unlike the <code>name</code> attribute of <code><xsl:variable></code> and <code><xsl:parameter></code> elements.</p> |
| Attributes | <p><code>\$name</code>—Defines the name of the parameter.</p> <p><code>value</code>—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.</p> |
| SLAX Example | <pre>param \$vrf; param \$dot = .;</pre> |
| XSLT Equivalent | <pre><xsl:parameter name="vrf"/> <xsl:parameter name="dot" select="."/></pre> |
| Usage Examples | See “Example: Requiring and Restricting Configuration Statements” on page 183, “Example: Imposing a Minimum MTU Setting” on page 189, “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Limiting the Number of ATM Virtual Circuits” on page 200, and “Example: Preventing Import of the Full Routing Table” on page 238. |
| Related Topics | ■ <code>var</code> on page 102 |

priority

| | |
|------------------------|---|
| Syntax | <code>priority <i>number</i>;</code> |
| Description | <p>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 <code>match</code> statement.</p> <p>This statement is comparable to the <code>priority</code> attribute of the <code><xsl:template></code> element. You can include this statement inside a SLAX <code>match</code> statement.</p> |
| SLAX Example | <pre>match * { priority 10; <output> .; }</pre> |
| XSLT Equivalent | <pre><xsl:template match="*" priority="10"> <output> <xsl:value-of select="."/> </output> </xsl:template></pre> |
| Usage Examples | None of the examples in this manual use this statement. |
| Related Topics | <ul style="list-style-type: none">■ apply-templates on page 91■ match on page 97 |

template

| | |
|------------------------|---|
| Syntax | <pre>template <i>qualified-name</i> (<i>parameter-name</i> = <i>value</i>) { /* code */ }</pre> |
| Description | <p>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.</p> |
| SLAX Example | <pre>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; } }</pre> |
| XSLT Equivalent | <pre><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></pre> |
| Usage Examples | <p>See all examples listed in “Commit Script Examples” on page 183.</p> |
| Related Topics | <ul style="list-style-type: none"> ■ call on page 92 ■ with on page 104 |

var

| | |
|------------------------|---|
| Syntax | <code>var \$name=value;</code> |
| Description | <p>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.</p> <p>In SLAX, parameter and variable names contain the dollar sign (\$) even in the declaration. This is unlike the name attribute of <code><xsl:variable></code> and <code><xsl:parameter></code> elements.</p> |
| Attributes | <p>\$name—Specifies the name of the variable. After declaration, the variable can be referred to within expressions using this name, including the \$ character.</p> <p>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.</p> |
| SLAX Example | <pre>var \$vrf; var \$location = \$dot/@location; var \$message = "We are in " _ \$location _ " now.";</pre> |
| XSLT Equivalent | <pre><xsl:variable name="vrf"/> <xsl:variable name="location" select="\$dot/location"/> <xsl:variable name="message" select="concat('We are in ', \$location, now.)"/></pre> |
| Usage Examples | See “Example: Limiting the Number of E1 Interfaces” on page 191, “Example: Limiting the Number of ATM Virtual Circuits” on page 200, “Example: Configuring Administrative Groups for LSPs” on page 230, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | ■ param on page 99 |

version

| | |
|------------------------|---|
| Syntax | version 1.0; |
| Description | <p>Specify the version of SLAX that is being used. All SLAX style sheets must begin with a <code>version</code> statement.</p> <p>Version 1.0 uses XML version 1.0 and XSLT version 1.1.</p> <p>In addition, the <code>xsl</code> namespace is implicitly defined as follows:</p> <pre>xmlns:xsl="http://www.w3.org/1999/XSL/Transform"</pre> |
| Attributes | <i>version-number</i> —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 183. |

with

| | |
|------------------------|---|
| Syntax | <code>with <i>name</i> = <i>value</i>;</code> |
| Description | <p>Specify a variable or parameter to be passed into a template. You can use this statement when you apply templates with the <code>apply-templates</code> statement or call templates with the <code>match</code> statement.</p> <p>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.</p> |
| Attributes | <p><i>name</i>—Name of the variable or parameter for which the value is being passed.</p> <p><i>value</i>—Value of the parameter being passed to the template.</p> |
| SLAX Example | <pre> 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 _ "); } </pre> |
| XSLT Equivalent | <pre> <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> </pre> |
| Usage Examples | See “Example: Configuring Dual Routing Engines” on page 234, “Example: Preventing Import of the Full Routing Table” on page 238, and “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241. |
| Related Topics | <ul style="list-style-type: none"> ■ apply-templates on page 91 ■ match on page 97 |

Part 2

Commit Scripts

- Commit Scripts Overview on page 107
- Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message on page 123
- Writing Commit Scripts That Generate a Persistent or Transient Configuration Change on page 137
- Writing Commit Scripts That Create Custom Configuration Syntax with Macros on page 153
- Configuring and Troubleshooting Commit Scripts on page 167
- Commit Script Examples on page 183
- Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts on page 269
- Summary of Commit Script Configuration Statements on page 277

Chapter 9

Commit Scripts Overview

This chapter includes the following topics:

- Commit Scripts Overview on page 107
- Advantages of Using Commit Scripts on page 108
- How Commit Scripts Work on page 109
- Required Boilerplate for Commit Scripts on page 115
- Design Considerations for Commit Scripts on page 117
- Line-by-Line Explanation of Sample Commit Scripts on page 119

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 two application programming interfaces (APIs) to the JUNOS Software: the JUNOS XML API and the JUNOScript API, which are discussed in “JUNOScript API and JUNOS XML API Overview” 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 “XSLT Overview” on page 15. For more information on SLAX, see “SLAX Overview” on page 27.

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 LDP-enabled interfaces are configured for an IGP, or ensure that all IGP-enabled interfaces are configured for LDP.
- IGP design check—Ensure that Level 1 IS-IS routers are never enabled.

When a candidate configuration does not adhere to your design rules, a commit script can instruct 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 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 183.



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 “Overview of Generating Persistent or Transient Configuration Changes” on page 137.

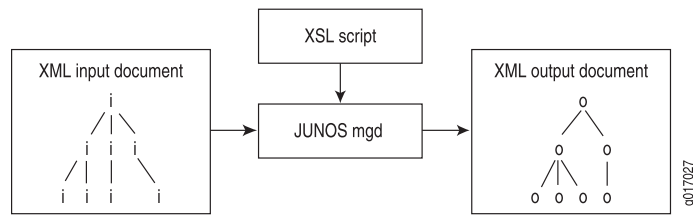
How Commit Scripts Work

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

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

Figure 4 on page 110 shows the flow of commit script input and output.

Figure 4: Commit Script Input and Output

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

- Commit Script Input on page 110
- Commit Script Output on page 111
- Commit Scripts and the JUNOS Software Commit Model on page 112
- Avoiding Potential Conflicts When Using Multiple Commit Scripts on page 114

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 CLI User Guide*.

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

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

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

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

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

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

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

By default, the file is placed in your home directory on the 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 52.

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 router. When the changes are complete, you can commit the configuration. The commit operation saves the candidate configuration changes into the current configuration.

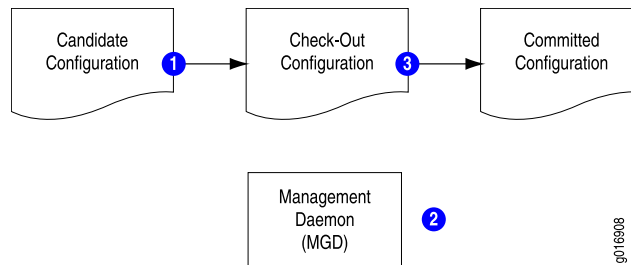
When you commit a set of changes in the candidate configuration, two methods are used to forward these changes to the current configuration:

- 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 management (mgd) process validates the candidate configuration based on standard JUNOS validation rules. If the configuration file is valid, it becomes the current active configuration. Figure 5 on page 112 and the accompanying discussion explain how the standard commit model works:

Figure 5: Standard Commit Model



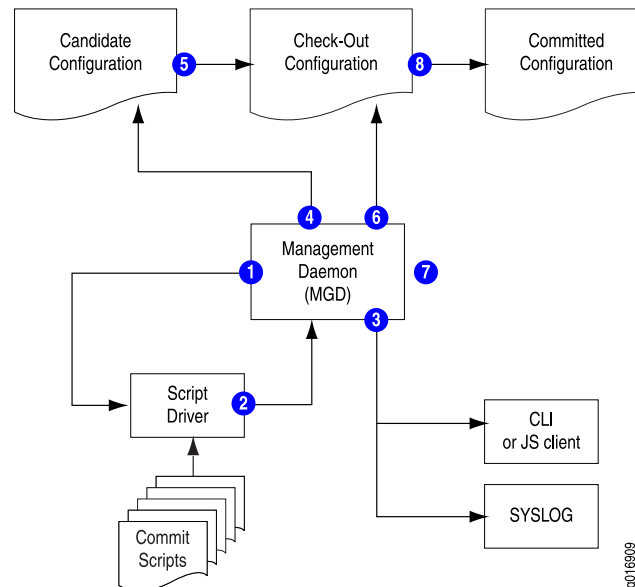
In the standard commit model, the software performs the following steps:

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

Commit Model with Commit Scripts

When commit scripts are added to the standard commit model, the process becomes more complex. The mgd process first passes an XML-formatted checkout configuration to a script driver, which handles the verification of the checkout configuration by the commit scripts. When verification is complete, the script driver returns an XML *action file* to the mgd process. The mgd process follows the instructions in the action file to update the candidate and checkout configurations, issue messages to the CLI, and write information to the system log as required. After processing the action file, the mgd process performs the standard JUNOS validation. Figure 6 on page 113 and the accompanying discussion explain this process.

Figure 6: 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 process sends the XML-formatted candidate configuration to the script driver.
2. Each enabled commit script is invoked against the candidate configuration, and each script can generate a set of actions for the mgd process to perform. The action are collected in an XML action file.
3. The mgd process performs the following actions in response to `<error>`, `<warning>`, and `<syslog>` tag elements in the action file:
 - `<error>`—The mgd process halts the commit process (that is, the commit operation fails), returns an error message to the CLI or JUNOScript client, and takes no further action.
 - `<warning>`—The mgd process forwards the message to the CLI or the JUNOScript client.
 - `<syslog>`—The mgd process forwards the message to the system log process.

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

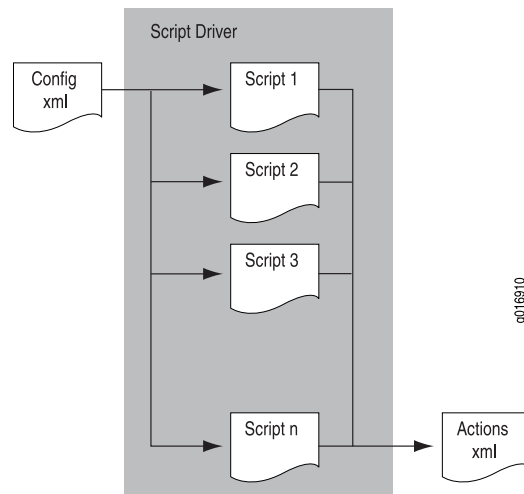
Changes made to the candidate configuration during the commit operation are not evaluated by the custom rules during that commit operation. However, persistent changes are maintained in the candidate configuration and are evaluated by the custom rules during subsequent commit operations. For more information about how commit scripts change the candidate configuration, see “Avoiding Potential Conflicts When Using Multiple Commit Scripts” on page 114.

Transient changes are never evaluated by the custom rules in commit scripts, because they are made to the checkout configuration only after the commit scripts have evaluated the candidate configuration and the candidate is copied to become the checkout configuration. To remove a transient change from the configuration, remove, disable, or deactivate the commit script (as discussed in “Controlling Execution of Commit Scripts During Commit Operations” on page 168), or comment out the code that generates the transient change.

For more information about differences between persistent and transient changes, see “Overview of Generating Persistent or Transient Configuration Changes” on page 137.

Avoiding Potential Conflicts When Using Multiple Commit Scripts

When you use multiple commit scripts, each script evaluates the original candidate configuration file. Changes made by one script are not evaluated by the other scripts. This means that conflicts between scripts might not be resolved when the scripts are first applied to the configuration. The commit scripts are executed in the order they are listed at the `[edit system scripts commit]` hierarchy level, as illustrated in Figure 7 on page 115.

Figure 7: Configuration Evaluation by Multiple Commit Scripts

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

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

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.

Required Boilerplate for Commit Scripts

When you write commit scripts, you use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided with the JUNOS Software. These tools include basic boilerplate that you must include in all commit scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make commit scripts easier to read and write, which you import from a file called **junos.xsl**. For more information about the extension functions and templates, see “JUNOS Extension Functions Overview” on page 39.

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.

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

**XSLT Boilerplate for
Commit Scripts**

```

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

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

```

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

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

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

```

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

```

Line 7 is an XSLT import statement. It loads the templates and variables from the file referenced as `../import/junos.xsl`, which ships as part of the JUNOS Software. The `junos.xsl` file contains a set of named templates you can call in your scripts. These named templates are discussed in “Templates in the `junos.xsl` File” on page 51.

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

Line 8 defines a template that matches the `<configuration>` element, which is the node selected by the `<xsl:template match="/">` template, contained in the `junos.xsl` import file. The `<xsl:template match="configuration">` element allows you to exclude the `/configuration/` root element from all XML Path Language (XPath) expressions in the script and begin XPath expressions with the top JUNOS hierarchy level. For more information, see “XPath Overview” on page 17.

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

Add your code between Lines 8 and 9.

Line 9 closes the template.

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

Line 10 closes the style sheet and the commit script.

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

SLAX Boilerplate for Commit Scripts

The corresponding SLAX boilerplate is as follows:

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

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

Design Considerations for Commit Scripts

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 provides some advice and common patterns for developing commit scripts.

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

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

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

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

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

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

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

```

<xsl:choose>
  <xsl:when test="$mpls/path-mtu/allow-fragmentation">
    <!-- ... -->
  </xsl:when>
  <xsl:when test="$mpls/hop-limit > 40">
    <!-- ... -->
  </xsl:when>
</xsl:choose>

```

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

```

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

```

If you channelize a cstm1-0/1/0 interface into 17 E1 interfaces, the script causes the following error message to appear when you issue the `commit` command. (For more information about this example, see “Example: Limiting the Number of E1 Interfaces” on page 191.)

```

[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

```


Line-by-Line Explanation of Sample Commit Scripts

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

Applying a Change to SONET/SDH Interfaces

The following commit script applies a transient change to each interface whose name begins with **so-**, setting the encapsulation to **ppp**. For information about transient changes, see “Overview of Generating Persistent or Transient Configuration Changes” on page 137. For a SLAX version of this example, see “Example: Generating a Transient Change” on page 150.

```

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.xml"/>

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

```

Lines 1 through 8 are boilerplate as described in “Required Boilerplate for Commit Scripts” on page 115 and are omitted here for brevity.

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

```

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

```

Line 10 is the open tag for a transient change. The possible contents of the `<transient-change>` element are the same as the contents of the `<configuration>` tag element in the 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>

```

Applying a Change to ISO-Enabled Interfaces

The following sample script ensures that interfaces that are enabled for an International Organization for Standardization (ISO) protocol also have MPLS enabled and are included at the [edit protocols mpls interface] hierarchy level. For a SLAX version of this example, see “Example: Controlling IS-IS and MPLS Interfaces” on page 203.

```

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.xml"/>

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

```

```

29         <xsl:text>Adding ISO-enabled interface </xsl:text>
30         <xsl:value-of select="$ifname"/>
31         <xsl:text> to [protocols mpls]</xsl:text>
32     </xsl:with-param>
33     <xsl:with-param name="dot" select="$mpls"/>
34     <xsl:with-param name="content">
35         <interface>
36             <name>
37                 <xsl:value-of select="$ifname"/>
38             </name>
39         </interface>
40     </xsl:with-param>
41 </xsl:call-template>
42 </xsl:if>
43 </xsl:for-each>
44 </xsl:template>
45 </xsl:stylesheet>

```

Lines 1 through 8 are boilerplate as described in “Required Boilerplate for Commit Scripts” on page 115 and are omitted here for brevity.

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

```

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

```

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

```

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

```

Line 11 builds the interface name in a variable. First, the **name** attribute of the variable declaration is set to **ifname**. In 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.xml** file. This template is discussed in “Importing the junos.xml File” on page 51.

```

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

Writing Commit Scripts That Generate a Custom Warning, Error, or System Log Message

This chapter discusses the following topics:

- Overview of Generating Custom Warning, Error, and System Log Messages on page 123
- Generating a Custom Warning, Error, or System Log Message on page 124
- Tag Elements to Use When Generating Messages on page 127
- Examples: Generating Custom Warning, Error, and System Log Messages on page 129

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

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

In the JUNOS 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 “Overview of Generating Persistent or Transient Configuration Changes” on page 137.

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 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.

Generating a Custom Warning, Error, or System Log Message

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

1. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from “Required Boilerplate for Commit Scripts” on page 115. It is reproduced here for convenience:

XSLT Boilerplate

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

  <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.xml";

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

```

2. At the position indicated by the comment “*Insert your code here*,” include one or more XSLT programming instructions or their SLAX equivalents. Commonly used XSLT constructs include the following. For detailed information, see “Summary of XPath and XSLT Constructs” on page 59 and “Summary of SLAX Statements” on page 91.)

- **<xsl:choose>** **<xsl:when>** **<xsl:otherwise>**—Conditional construct that causes different instructions to be processed in different circumstances. The **<xsl:choose>** instruction contains one or more **<xsl:when>** elements, each of which tests an XPath expression. If the test evaluates as true, the XSLT processor executes the instructions in the **<xsl:when>** element. The XSLT processor processes only the instructions contained in the first **<xsl:when>** element whose **test** attribute evaluates as true. If none of the **<xsl:when>** elements’ **test** attributes evaluate as true, the content of the **<xsl:otherwise>** element, if there is one, is processed.
- **<xsl:for-each select=“xpath-expression”>**—Programming instruction that tells the XSLT processor to gather together a set of nodes and process them one by one. The nodes are selected by the Extensible Markup Language (XML) Path Language (XPath) expression in the **select** attribute. Each of the nodes is then processed according to the instructions contained in the **<xsl:for-each>** instruction. Code inside an **<xsl:for-each>** instruction is evaluated recursively for each node that matches the XPath expression. The context is moved to the node during each pass.
- **<xsl:if test=“xpath-expression”>**—Conditional construct that causes instructions to be processed if the XPath expression in the **test** attribute evaluates to **true**.

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

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

In SLAX, the if construct looks like this:

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

For more information about how to use programming instructions, including examples and pseudocode, see “XSLT Programming Instructions Overview” on page 22. For information about writing scripts in SLAX instead of XSLT, see “SLAX Overview” on page 27.

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

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

This `<xnm:warning>` element:

```
"<xnm:warning>" on page 275
<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>" on page 273
<xsl:call-template name="jcs:edit-path"/>
<xsl:call-template name="jcs:statement"/>
<message>Missing a description for this T1 interface.</message>
</xnm:error>
```

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

```
[edit]

user@host# commit

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



NOTE: If you are including a warning message in conjunction with a script-generated configuration change, you can generate the warning by including the `message` parameter with the `<jcs:emit-change>` template. The `message` parameter causes the `<jcs:emit-change>` template to call the `<xnm:warning>` template, which sends a warning notification to the CLI. (For more information, see “Overview of Generating Persistent or Transient Configuration Changes” on page 137.)

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

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

For a description of all the XSLT tags and attributes you can include, see “Tag Elements to Use When Generating Messages” on page 127.

For SLAX versions of these constructs, see “Example: Generating a Custom Warning Message” on page 129, “Example: Generating a Custom Error Message” on page 132, and “Example: Generating a Custom System Log Message” on page 135.

- 4. Save the script with a meaningful name.
- 5. Copy the script to either the /var/db/scripts/commit directory on the hard drive or the /config/scripts/commit directory on the flash drive. For information on setting the storage location for commit scripts, see “Storing Commit Scripts in Flash Memory” on page 171.

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

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

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

where filename is the name that you previously assigned to the script.

Tag Elements to Use When Generating Messages

Table 8 on page 127 summarizes the tag elements that you can include in a custom warning, error, or system log message. For examples of how to supply data values within a script, see “Examples: Generating Custom Warning, Error, and System Log Messages” on page 129 and “Commit Script Examples” on page 183. (For detailed information about element hierarchy, see “Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts” on page 269.)

Table 8: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages

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

Table 8: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages (continued)

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

Table 8: Tags and Attributes for Creating Custom Warning, Error, and System Log Messages (continued)

| Data Item, XML Element, or Attribute | Required or Supported | Description |
|--|--|---|
| <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 | <p>Emits an <edit-path> element, which specifies the CLI configuration mode edit path in effect when the warning or error was generated.</p> <p>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].</p> |
| <xsl:call-template name="jcs:statement"> | Supported in warning and error messages only | <p>Emits a <statement> element, which describes the configuration statement in effect when the warning or error was generated.</p> <p>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.</p> |

Examples: Generating Custom Warning, Error, and System Log Messages

- Example: Generating a Custom Warning Message on page 129
- Example: Generating a Custom Error Message on page 132
- Example: Generating a Custom System Log Message on page 135

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 used in “Generating a Custom Warning, Error, or System Log Message” on page 124.)

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.xml"/>

  <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.xml";

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 Warning Message Generated by the Commit Script

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

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

1. Copy the XSLT script from the preceding section into a text file named `source-route.xml`.
2. Copy the `source-route.xml` file to the `/var/db/scripts/commit` directory on the router hard drive or the `/config/scripts/commit` directory on the flash drive.
3. Include the file `source-route.xml` statement at the `[edit system scripts commit]` hierarchy level:

```

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

```

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. The following output appears:

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

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

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

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

```
[edit]
user@host# commit check | display detail
2009-06-15 14:40:29 PDT: reading commit script configuration
2009-06-15 14:40:29 PDT: testing commit script configuration
2009-06-15 14:40:29 PDT: opening commit script
'/var/db/scripts/commit/source-route-warning.xml'
2009-06-15 14:40:29 PDT: reading commit script 'source-route-warning.xml'
2009-06-15 14:40:29 PDT: running commit script 'source-route-warning.xml'
2009-06-15 14:40:29 PDT: processing commit script 'source-route-warning.xml'
[edit chassis]
    warning: IP source-route processing is not enabled.
2009-06-15 14:40:29 PDT: no errors from source-route-warning.xml
2009-06-15 14:40:29 PDT: saving commit script changes
2009-06-15 14:40:29 PDT: summary: changes 0, transients 0 (allowed), syslog 0
2009-06-15 14:40:29 PDT: no commit script changes
2009-06-15 14:40:29 PDT: exporting juniper.conf
2009-06-15 14:40:29 PDT: expanding groups
2009-06-15 14:40:29 PDT: finished expanding groups
2009-06-15 14:40:29 PDT: setup foreign files
2009-06-15 14:40:29 PDT: propagating foreign files
2009-06-15 14:40:30 PDT: complete foreign files
2009-06-15 14:40:30 PDT: daemons checking new configuration
configuration check succeeds
```

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 | <pre> <?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> </pre> |
| SLAX Syntax | <pre> 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."; } } } } </pre> |

Verifying the Error Message Generated by the Commit Script

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

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

1. Copy the XSLT script from the preceding section into a text file named `description.xml`.
2. Copy the `description.xml` file to the `/var/db/scripts/commit` directory on the router hard drive or the `/config/scripts/commit` directory on the flash drive.
3. Include the `description.xml` statement at the `[edit system scripts commit]` hierarchy level:

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

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

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

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

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

```

        <statement>
            description
        </statement>
    </message>
    Missing a description for this T1 interface.
</message>
</xnm:error>
<xnm:error>
    <edit-path>
        [edit interfaces interface t1-0/0/2]
    </edit-path>
    <statement>
        description
    </statement>
    <message>
        Missing a description for this T1 interface.
    </message>
</xnm:error>
<xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm"
            xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
    <message>
        2 errors reported by commit scripts
    </message>
</xnm:error>
<xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm"
            xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
    <message>
        commit script failure
    </message>
</xnm:error>
</routing-engine>
</commit-results>
<cli>
    <banner>[edit interfaces]</banner>
</cli>
</rpc-reply>

```

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

```

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

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 that a commit script generates a system log message correctly, make sure that the candidate configuration contains the condition that elicits the system log message. 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 topic, perform the following steps:

1. Copy the XSLT script from the preceding section into a text file named `read-write.xml`.
2. Copy the `read-write.xml` file to the `/var/db/scripts/commit` directory on the router hard drive or the `/config/scripts/commit` directory on the flash drive.
3. Include the `file read-write.xml` statement at the `[edit system scripts commit]` hierarchy level:

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

4. 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
```

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

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

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

6. Issue the `commit` command:

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

When the commit operation completes, inspect the system log file. The default directory for log files is `/var/log`. System log entries generated by commit scripts have the following format:

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

For example:

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

Chapter 11

Writing Commit Scripts That Generate a Persistent or Transient Configuration Change

This chapter discusses the following topics:

- Overview of Generating Persistent or Transient Configuration Changes on page 137
- Generating a Persistent or Transient Change on page 141
- Removing a Persistent or Transient Change on page 145
- Tag Elements to Use When Generating Persistent and Transient Changes on page 147
- Examples: Generating Persistent and Transient Changes on page 148

Overview of Generating Persistent or Transient Configuration 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.

- Differences Between Persistent and Transient Changes on page 137
- Interaction of Configuration Changes and Configuration Groups on page 140
- Tag Elements and Templates for Generating Changes on page 140

Differences Between Persistent and Transient Changes

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

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

A transient change, in contrast, is made in the *checkout configuration* but not in the candidate configuration. The checkout configuration is the configuration database that is inspected for standard JUNOS syntax just before it is copied to become the active configuration on the router. If you subsequently remove or disable the commit

script that made the change and reissue the **commit** command, the change is no longer made to the checkout configuration and so does not affect the active configuration. In other words, removing the commit script effectively removes a transient change from the configuration.

A common use for transient changes is to eliminate the need to repeatedly configure and display well-known policies, thus allowing these policies to be enforced implicitly. For example, if MPLS must be enabled on every interface with an International Organization for Standardization (ISO) protocol enabled, the change can be transient, so that the repetitive or redundant configuration data need not be carried or displayed in the candidate configuration. Furthermore, transient changes allow you to write script instructions that apply the change only if a set of conditions is met.

Persistent and transient changes are loaded 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 9 on page 138.

Table 9: Differences Between Persistent and Transient Changes

| Persistent Changes | Transient Changes |
|--|---|
| A persistent change is represented in a commit script by the <code><change></code> tag. | A transient change is represented in a commit script by the <code><transient-change></code> tag. |
| Another way to represent a persistent change is with the <code>content</code> parameter inside a call to the <code><jcs:emit-change></code> template. | Another way to represent a transient change is to use the <code>content</code> parameter and the <code>tag transient</code> parameter inside a call to the <code><jcs:emit-change></code> template. |
| The <code><jcs:emit-change></code> template is a helper template contained in the <code>junos.xsl</code> import file. | |
| 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 <code>allow-transients</code> statement at the <code>[edit system scripts commit]</code> hierarchy level. If you enable a commit script that generates transient changes and you do not include the <code>allow-transients</code> 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 <code>allow-transients</code> statement at the <code>[edit system scripts commit]</code> hierarchy level. Rather, you must include this statement directly by using the CLI. |

Table 9: Differences Between Persistent and Transient Changes *(continued)*

| Persistent Changes | Transient Changes |
|--|---|
| Persistent changes work like the <code>load merge</code> command in that they cause the software to merge the incoming configuration into the current candidate configuration. | Transient changes work like the <code>load update</code> command in that they cause the software to replace only the configuration that has changed. |
| 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 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. | 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. |
| 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. | |
| After committing a script that causes a persistent change to be generated, you can view the persistent change by issuing the <code>show</code> configuration mode command: user@host# show | After committing a script that causes a transient change to be generated, you can view the transient change by issuing the <code>show display commit-scripts</code> configuration mode command: user@host# show display commit-scripts |
| This command displays persistent changes only, not transient changes. | This command displays both persistent and transient changes. |
| Persistent changes must conform to your custom configuration design rules as dictated by commit scripts. | 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 112. |
| 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. | |
| 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 <code>commit</code> 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 <code>commit</code> command. | You cannot remove a transient change by rolling back to a previous configuration. |

Table 9: Differences Between Persistent and Transient Changes (*continued*)

| Persistent Changes | Transient Changes |
|---|--|
| 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. |

Interaction of Configuration Changes and Configuration Groups

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

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



CAUTION: Each commit script inspects the postinheritance view of the configuration. If a candidate configuration contains a configuration group, be careful when using a commit script to change the related target configuration, because doing so might alter the intended inheritance from the configuration group.

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

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

Tag Elements and Templates for Generating Changes

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

The `<change>` and `<transient-change>` elements are similar to the `<load-configuration>` 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 used in the JUNOScript `<load-configuration>` operation. For complete details about the `<load-configuration>` element, see the *JUNOScript API Guide*.

Generating a Persistent or Transient Change

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

1. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from “Required Boilerplate for Commit Scripts” on page 115. It is reproduced here for convenience:

XSLT Boilerplate

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

  <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.xml";

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

2. At the position indicated by the comment “*Insert your code here*,” include one or more XSLT programming instructions or their SLAX equivalents. Commonly used XSLT constructs include the following. For detailed information, see “Summary of XPath and XSLT Constructs” on page 59 and “Summary of SLAX Statements” on page 91.)
 - `<xsl:choose>` `<xsl:when>` `<xsl:otherwise>`—Conditional construct that causes different instructions to be processed in different circumstances. The `<xsl:choose>` instruction contains one or more `<xsl:when>` elements, each of which tests an XPath expression. If the test evaluates as true, the XSLT processor executes the instructions in the `<xsl:when>` element. The XSLT processor processes only the instructions contained in the first `<xsl:when>` element whose `test` attribute evaluates as true. If none of the `<xsl:when>`

elements' **test** attributes evaluate as true, the content of the `<xsl:otherwise>` element, if there is one, is processed.

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

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

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

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

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

For more information about how to use programming instructions, including examples and pseudocode, see “XSLT Programming Instructions Overview” on page 22. For information about writing scripts in SLAX instead of XSLT, see “SLAX Overview” on page 27.

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 52.)

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

Persistent Change Generated with the <change> Element

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

This example includes the current hierarchy information in the warning message by referencing the `<jcs:edit-path>` and `<jcs:statement>` templates. For more information about warning messages, see “Overview of Generating Custom Warning, Error, and System Log Messages” on page 123.

```
<xsl:for-each select="interfaces/interface[starts-with(name, 'so-')]">
  <xsl:if test="not(unit/family/mps)">
    <change>
      <interfaces>
        <interface>
          <name><xsl:value-of select="name"/></name>
          <unit>
            <name><xsl:value-of select="unit/name"/></name>
            <family>
              <mps/>
            </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 52.)

```

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

```

Transient Change Generated with the <transient-change> Element

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

```

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

```

4. Save the script with a meaningful name.
5. Copy the script to either the /var/db/scripts/commit directory on the hard drive or the /config/scripts/commit directory on the flash drive. For information on setting the storage location for commit scripts, see "Storing Commit Scripts in Flash Memory" on page 171.

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

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

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

where *filename* is the name you assigned in 4.

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

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

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

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

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

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

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

Persistent 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.

Removing a Persistent or Transient Change

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

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

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

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

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

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

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

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

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

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

3. Issue the commit command:

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

4. If you are deleting the reference to the script from the configuration, you can also remove the file from commit scripts storage directory (either `/var/db/scripts/commit` on the hard drive or `/config/scripts/commit` on the flash drive; for information on setting the storage location for commit scripts, see “Storing Commit Scripts in Flash Memory” on page 171.) To do this, exit configuration mode and issue the **file delete** operational mode command:

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

```
user@host> file delete /var/db/scripts/commit/filename
```

- OR -

```
user@host> file delete /config/scripts/commit/filename
```

Tag Elements to Use When Generating Persistent and Transient Changes

Table 10 on page 147 describes the data that you can include in the `<change>` tag element in a commit script. To see how data values are supplied within a script, see “Examples: Generating Persistent and Transient Changes” on page 148 and “Commit Script Examples” on page 183. (For detailed information about element hierarchy, see “Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts” on page 269.)

Table 10: Tags and Attributes for Creating Configuration Changes

| Data Item, XML Element, or Attribute | Description |
|---|--|
| Container Tags | |
| <code><change></code> | Request that the JUNOScript server load configuration data into the candidate configuration. |
| <code><change [action="action"]></code> | Set the <code>action</code> attribute to <code>merge</code> , <code>override</code> , <code>replace</code> , or <code>update</code> . By default, the action is <code>merge</code> for persistent changes and <code>update</code> for transient changes. The data enclosed in the <code><change></code> tag must be JUNOS XML tag elements only. |
| <code><change rollback="index"/></code> | Set the <code>rollback</code> 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. |
| <code><change url="url" [action="action"]></code> | Set the <code>url</code> attribute to the pathname of a file that resides on the routing platform and contains the configuration data to load. The action can be <code>merge</code> , <code>override</code> , <code>replace</code> , or <code>update</code> . By default, the action is <code>merge</code> for persistent changes and <code>update</code> for transient changes. The data must be JUNOS XML tag elements only. |
| <code><configuration></code> | Enclose JUNOS XML and JUNOScript tag elements in a <code><configuration></code> 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 “Required Boilerplate for Commit Scripts” on page 115. |
| <code><transient-change></code> | Request that the JUNOScript server load configuration data into the configuration. |
| Content Tags | |
| <code><jcs:emit-change></code> | This is a template in the file <code>junos.xsl</code> . This template converts the contents of the <code><xsl:with-param></code> element into a <code><change></code> request. |

Table 10: Tags and Attributes for Creating Configuration Changes *(continued)*

| Data Item, XML Element, or Attribute | Description |
|---|--|
| <code><xsl:with-param name="content"></code> | You use the <code>content</code> parameter with the <code><jcs:emit-change></code> template. Allows you to include the content of the change, relative to <code>dot</code> . |
| <code><xsl:with-param name="tag" select="'transient-change'"/></code> | <p>Convert the contents of the <code>content</code> parameter into a <code><transient-change></code> request.</p> <p>You use the <code>tag</code> parameter with the <code><jcs:emit-change></code> template.</p> <p>By default, the <code><jcs:emit-change></code> template converts the contents of the <code>content</code> parameter into a <code><change></code> (persistent change) request.</p> |

Examples: Generating Persistent and Transient Changes

This section is organized as follows:

- Example: Generating a Persistent Change on page 148
- Example: Generating a Transient Change on page 150

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.xml` 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.xml"/>

  <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.xml";

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 Persistent Change Generated by the Commit Script

To test that a commit script generates a persistent change correctly, make sure that the candidate configuration contains the condition that elicits the change. 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.

The sample script produces a message announcing the change it is making. To display the XML-formatted version of the message, issue the `commit check | display xml` command:

```

[edit]
user@host# commit check | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
  <commit-results>
    <routing-engine junos:style="normal">
      <name>re0</name>
      <xnm:warning>
        <edit-path>

```

```

[edit 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>
@@@&lt;/rpc-reply>

```

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

```

[edit]
user@host# commit check | display detail
2009-06-17 14:17:35 PDT: reading commit script configuration
2009-06-17 14:17:35 PDT: testing commit script configuration
2009-06-17 14:17:35 PDT: opening commit script '/var/db/scripts/commit/mps.xml'
2009-06-17 14:17:35 PDT: reading commit script 'mps.xml'
2009-06-17 14:17:35 PDT: running commit script 'mps.xml'
2009-06-17 14:17:35 PDT: processing commit script 'mps.xml'
2009-06-17 14:17:35 PDT: no errors from mps.xml
2009-06-17 14:17:35 PDT: saving commit script changes
2009-06-17 14:17:35 PDT: summary: changes 0, transients 0 (allowed), syslog 0
2009-06-17 14:17:35 PDT: no commit script changes
2009-06-17 14:17:35 PDT: finished loading commit script changes
2009-06-17 14:17:35 PDT: exporting juniper.conf
2009-06-17 14:17:35 PDT: expanding groups
2009-06-17 14:17:35 PDT: finished expanding groups
2009-06-17 14:17:35 PDT: setup foreign files
2009-06-17 14:17:35 PDT: propagating foreign files
2009-06-17 14:17:35 PDT: complete foreign files
2009-06-17 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 | <pre> <?xml version="1.0" standalone="yes"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" </pre> |
|--------------------|---|


```

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 Transient Change Generated by the Commit Script

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

```

[edit]
user@host# commit check | display detail
2009-06-15 12:07:30 PDT: reading commit script configuration
2009-06-15 12:07:30 PDT: testing commit script configuration
2009-06-15 12:07:30 PDT: opening commit script
'/var/db/scripts/commit/transient.xsl'
2009-06-15 12:07:30 PDT: reading commit script 'transient.xsl'
2009-06-15 12:07:30 PDT: running commit script 'transient.xsl'
2009-06-15 12:07:30 PDT: processing commit script 'transient.xsl'
2009-06-15 12:07:30 PDT: no errors from transient.xsl
2009-06-15 12:07:30 PDT: saving commit script changes
2009-06-15 12:07:30 PDT: summary: changes 0, transients 2 (allowed), syslog 0
2009-06-15 12:07:30 PDT: no commit script changes
2009-06-15 12:07:30 PDT: exporting juniper.conf
2009-06-15 12:07:30 PDT: loading transient changes
2009-06-15 12:07:30 PDT: loading commit script changes(transient)

```

```

2009-06-15 12:07:30 PDT: finished loading commit script changes
2009-06-15 12:07:30 PDT: expanding groups
2009-06-15 12:07:30 PDT: finished expanding groups
2009-06-15 12:07:30 PDT: setup foreign files
2009-06-15 12:07:30 PDT: propagating foreign files
2009-06-15 12:07:31 PDT: complete foreign files
2009-06-15 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 ...

```

Chapter 12

Writing Commit Scripts That Create Custom Configuration Syntax with Macros

This chapter discusses the following topics:

- Overview of Creating Custom Configuration Syntax with Macros on page 153
- How Macros Work on page 153
- Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 159
- Example: Creating Custom Configuration Syntax with Macros on page 161

Overview of Creating Custom Configuration Syntax with Macros

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

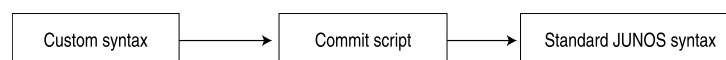
Commit scripts generally impose restrictions on JUNOS Software configuration and automatically correct configuration mistakes when they occur (as discussed in “Overview of Generating Persistent or Transient Configuration Changes” on page 137). However, macros are useful for an entirely different reason. Commit scripts that contain macros do not generally correct configuration mistakes, nor do they necessarily restrict configuration. Instead, they provide a way to simplify and speed configuration tasks, thereby preventing mistakes from occurring at all.

For a detailed example of how macros can save time and effort, see “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241.

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 153. The standard JUNOS statements are added to the configuration to cause your intended operational changes.

Figure 8: Macro Input and Output



9016782

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 154
- `<data>` Element on page 155
- Expanding the Custom Syntax on page 156
- Other Ways to Use Macros on page 159

Creating a Custom Syntax

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

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

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

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

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

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

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

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

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

The resulting configuration statements are as follows:

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

<data> Element

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

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

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

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

```

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

```

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

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

Expanding the Custom Syntax

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

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

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

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

The SLAX equivalent is:

```
for-each (protocols/mppls/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>
```

SLAX uses the underscore (`_`) to concatenate values:

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

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

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

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

The SLAX equivalent is:

```
<transient-change> {
  <protocols> {
    <mpls> {
```

```

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

```



NOTE: The example shown here is partial. For a full example, see “Example: Creating Custom Configuration Syntax with Macros” on page 161.

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

```

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

```

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

Other Ways to Use Macros

The example discussed in “Creating a Custom Syntax” on page 154 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 154, you can also use macros to prevent a commit script from performing a task. For example, a basic commit script that automatically adds MPLS configuration to interfaces can make an exception for interfaces you explicitly tag as not requiring MPLS, by testing for the presence of an **apply-macro** statement named **no-mpls**. For an example of this use of macros, see “Example: Controlling LDP Configuration” on page 212.

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 router. 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. At the start of the script, include the Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) boilerplate from “Required Boilerplate for Commit Scripts” on page 115. It is reproduced here for convenience:

XSLT Boilerplate

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

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

SLAX Boilerplate

```

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

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

```

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

For detailed information about XSLT and SLAX constructs, see “Summary of XPath and XSLT Constructs” on page 59 and “Summary of SLAX Statements” on page 91.)

For an example that uses both types of instructions and includes a line-by-line analysis of the XSLT syntax, see “Example: Creating Custom Configuration Syntax with Macros” on page 161.

3. Save the script with a meaningful name.
4. Copy the script to either the `/var/db/scripts/commit` directory on the hard drive or the `/config/scripts/commit` directory on the flash drive. For information on setting the storage location for commit scripts, see “Storing Commit Scripts in Flash Memory” on page 171.

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

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

```

[edit system scripts commit]
file filename;

```

where *filename* is the name you previously assigned to the script.

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

```

[edit system scripts commit]
allow-transients;

```

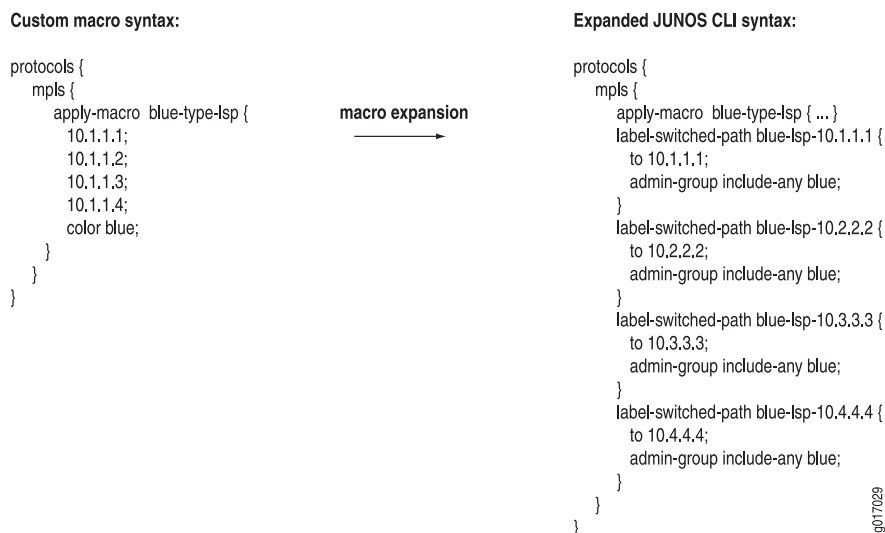
If all the commit scripts run without errors, any transient changes are loaded into the checkout configuration, but not to the candidate configuration. Any persistent changes are loaded into the candidate configuration. The commit process then

continues by validating the configuration and propagating changes to the affected processes on the router.

Example: Creating Custom Configuration Syntax with Macros

Figure 9 on page 161 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



In this example, the JUNOS management (mgd) process inspects the configuration, looking for `apply-macro` statements. For each `apply-macro` statement with the `color` parameter included at the `[edit protocols mpls]` hierarchy level, the script generates a transient change, using the data provided within the `apply-macro` statement to expand the macro into a standard JUNOS 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 161 and a line-by-line explanation of the script:

| | |
|--------------------|--|
| XSLT Syntax | <pre> 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']"> </pre> |
|--------------------|--|

```

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

```

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

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

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

Line 10 selects every **apply-macro** statement at the [edit protocols mpls] hierarchy level that contains the **color** parameter. The sample configuration in Figure 9 on page 161 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 161 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.xml";

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

```

    <admin-group> {
        <include-any> $color;
    }
}
}
}
}
}
}
}
}
}
}

```

For more information about this example, see “Example: Configuring Administrative Groups for LSPs” on page 230.

Verifying the Configuration Statements Generated by the Commit Script

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

```

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

```


Chapter 13

Configuring and Troubleshooting Commit Scripts

At commit time, the JUNOS management process (mgd) looks in the `/config/scripts/commit` or the `/var/db/scripts/commit` directory, depending on whether the scripts are stored on the flash drive or the hard drive, for one or more commit scripts. Each commit script executes against the candidate configuration database to ensure the configuration conforms to the rules dictated by the scripts.

This chapter discusses the following topics:

- Implementing Commit Scripts on page 168
- Controlling Execution of Commit Scripts During Commit Operations on page 168
- Storing Commit Scripts in Flash Memory on page 171
- Overview of Updating Commit Scripts from a Remote Source on page 171
- Configuring the Master Source for a Commit Script on page 173
- Updating a Commit Script from the Master Source on page 173
- Updating a Commit Script from an Alternate Location on page 174
- Executing Large Commit Scripts on page 174
- Converting a Commit Script from XSLT to SLAX on page 175
- Converting a Commit Script from SLAX to XSLT on page 175
- Displaying Commit Script Output on page 176
- Tracing Commit Script Processing on page 177
- Troubleshooting Commit Scripts on page 181

Implementing Commit Scripts

To use a commit script on a router or switch, follow these steps:

1. Write the commit script.

For information about the types of commit scripts you can write, see “Overview of Generating Custom Warning, Error, and System Log Messages” on page 123, “Overview of Generating Persistent or Transient Configuration Changes” on page 137, and “Overview of Creating Custom Configuration Syntax with Macros” on page 153. For examples, see “Commit Script Examples” on page 183.

2. Copy the script to the `/var/db/scripts/commit` directory on the hard drive or the `/config/scripts/commit` directory on the flash drive; for information about setting the storage location for scripts, see “Storing Commit Scripts in Flash Memory” on page 171. Only users who belong to the JUNOS **super-user** login class can access and edit files in these directories.



NOTE: If the router or switch has dual Routing Engines and you want to enable the commit script to execute on both Routing Engines, you must copy it to the `/var/db/scripts/commit` or `/config/scripts/commit` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

3. Enable the script by including the file *filename* statement at the `[edit system scripts commit]` hierarchy level. For instructions, see “Controlling Execution of Commit Scripts During Commit Operations” on page 168.
4. Issue the `commit` command.

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

Controlling Execution of Commit Scripts During Commit Operations

Commit scripts are stored on a router or switch’s hard drive (in the `/var/db/scripts/commit` directory) or flash drive (in the `/config/scripts/commit` directory); for more information, see “Storing Commit Scripts in Flash Memory” on page 171. However, a commit script is not actually executed during commit operations unless its filename is included at the `[edit system scripts commit file]` hierarchy level. To prevent execution of a commit script, delete the commit script’s filename at that hierarchy level.

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

want to remove the commit script filename at the [edit system scripts commit file] hierarchy level. To enable use of the script again later, you simply replace the file in the commit script directory.



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

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

To determine which commit scripts are currently active on the router, either list the contents of the `/var/run/scripts/commit` directory or use the **show** command to display the files included (but not marked inactive:) at the [edit system scripts commit] hierarchy level.

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

See the following sections:

- Enabling Commit Scripts to Execute During Commit Operations on page 169
- Preventing Commit Scripts from Executing During Commit Operations on page 170
- Deactivating Commit Scripts on page 170
- Activating Commit Scripts on page 171

Enabling Commit Scripts to Execute During Commit Operations

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

1. Ensure that the commit script is located in the correct directory: the `/var/db/scripts/commit` directory on the hard drive or the `/config/scripts/commit` directory on the flash drive. For more information about script storage location, see “Storing Commit Scripts in Flash Memory” on page 171.
2. Enable the commit script by including the **file filename** statement at the [edit system scripts commit] hierarchy level. Only users who belong to the JUNOS super-user login class can enable commit scripts.

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

- *filename*—Name of the commit script.

- **optional**—Enable the commit operation to succeed when the script file does not exist in the script directory. If this statement is omitted, the commit operation fails if the script does not exist.
3. Commit the configuration:


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

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

Preventing Commit Scripts from Executing During Commit Operations

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

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


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

filename—Name of the commit script.
2. Remove the commit script from the commit script directory. Although removing the commit script from the commit script directory is not necessary, it is always a good policy to delete unused files from the system.
3. Commit your changes:


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

Deactivating Commit Scripts

To deactivate a commit script, follow these steps:

1. Issue the **deactivate** command:


```
[edit]
user@host# deactivate system scripts commit file filename
```
2. Commit your changes:


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

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

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

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

```
user@host# show system scripts commit
inactive: file mycommit.slax
```

Activating Commit Scripts

To activate an inactive commit script, follow these steps:

1. Issue the **activate** command:

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

2. Commit your changes:

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

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

Storing Commit Scripts in Flash Memory

By default, commit scripts are stored in the `/var/db/scripts/commit` directory on the router's hard drive. To store them in flash memory instead, include 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; commit scripts are stored in the `/config/scripts/commit` directory on the flash drive. Changing the scripts' physical location has no effect on their operation.



NOTE: When you add or remove the `load-scripts-from-flash` statement in the configuration, you must manually move scripts from the hard drive to the flash drive, or vice versa, as appropriate. They are not moved automatically.

Overview of Updating Commit Scripts from a Remote Source

You can update the commit scripts on a router by retrieving a copy of them from a remote machine (which can be another router or a regular networked computer). This eases file management, because it enables you to update a script in a single location and have routers update their copies from that location. Each router continues to use its locally stored commit scripts during commit operations, only updating a script when you issue the appropriate configuration mode command.

For each commit script, you can define a remote location that houses the master copy of the script, by specifying its URL with the `source` statement at the `[edit system scripts commit file filename]` hierarchy level. When you then issue the `set refresh` configuration mode command for a script, the router updates its local copy by retrieving the remote master copy from that URL.

You can also store a copy of a particular script at a remote location other than the master source. This is convenient when, for example, the master source cannot be accessed due to network or other problems. When you issue the **set refresh-from** command for the script, you specify the URL for the remote script as an option to the command.

You can use the **set refresh** and **set refresh-from** commands to update either an individual commit script or all enabled commit scripts on the router. When you issue the **set refresh** or **set refresh-from** command, the router immediately attempts to connect to the appropriate remote source for each script. If successful, the router updates the local commit script with the remote source. If a problem occurs, a set of error messages is returned.

Issuing the **set refresh** or **set refresh-from** command does not add the **refresh** and **refresh-from** statements to the configuration. In other words, the **set** command behaves differently for these statements than for others: it behaves like an operational mode command by executing an operation, instead of adding a statement to the configuration.

If a router has dual Routing Engines and you want the script to be updated on both Routing Engines, you must include the **refresh** or **refresh-from** statements in the configuration of both Routing Engines. The **commit synchronize** command does not cause the **refresh** or **refresh-from** statement to update scripts on both Routing Engines.

The **refresh** and **refresh-from** statements are mutually exclusive.



CAUTION: We recommend that you do not automate the update function by including the **refresh** statement as a commit script change element. Even though this might seem like a good way to ensure that the most current commit script is always used, we recommend against it for the following reasons:

- Automated update means that the network must be operational for the commit operation to succeed. If the network goes down after you make a configuration error, you cannot recover quickly.
 - If multiple commit scripts need to be update during each commit operation, the network response time can slow down.
 - Automated update is always the last action performed during a commit operation. Consequently, the updated commit script executes only during the next commit operation. This is because commit scripts are applied to the candidate configuration before the software copies any persistent changes generated by the scripts to the candidate configuration. For more information, see “Commit Scripts and the JUNOS Software Commit Model” on page 112. In contrast, if you perform the update operation manually, the updated commit script takes effect as expected, that is, immediately after you commit the **refresh** statement in the configuration.
 - If you automate the update operation, the **refresh-from** statement has no effect, because the **refresh-from** URL conflicts with and is overridden by the **source** statement URL. For information about the **refresh-from** statement, see “Updating a Commit Script from an Alternate Location” on page 174.
-

- Related Topics**
- Configuring the Master Source for a Commit Script on page 173
 - Updating a Commit Script from the Master Source on page 173
 - Updating a Commit Script from an Alternate Location on page 174

Configuring the Master Source for a Commit Script

You can store a master copy of each commit script in a central repository. This eases file management because you can make changes to the master commit script in one place and then update the copy on each router where the commit script is currently enabled.

To specify the location of the master source for a commit script, include the **source** statement at the `[edit system scripts commit file filename]` hierarchy level:

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

- *filename*—Name of the commit script.
- *url*—URL of the commit script's master source file. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

Including the **source** statement in the configuration does not affect the local copy of the commit script until you issue the **set refresh** command as described in “Updating a Commit Script from the Master Source” on page 173. At that point, the master copy is retrieved from the specified URL and overwrites the local copy.

Updating a Commit Script from the Master Source

To update a single commit script from its master source, issue the **set refresh** command at the `[edit system scripts commit file filename]` hierarchy level. The master source must already be configured as described in “Configuring the Master Source for a Commit Script” on page 173.

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

To update all enabled commit scripts from their master sources, issue the **set refresh** command at the `[edit system scripts commit]` hierarchy level:

```
[edit system scripts commit]
user@host# set refresh
```

When you issue the **set refresh** command, the router immediately attempts to connect to the device that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local commit scripts directory. The updated commit script is executed when you next issue the **commit** command. If no master source for a script is defined, it is not updated and a warning is issued.

If the router has dual Routing Engines and you want to update a script on both Routing Engines, you must issue the **set refresh** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh** statement to update scripts on both Routing Engines.

Updating a Commit Script from an Alternate Location

In addition to updating a commit script from the master source defined by the **source** statement at the [edit system scripts commit file *filename*] hierarchy level, you also can update a script from an alternate location. This is convenient when, for example, the master source cannot be accessed due to network or other problems. To update a single commit script from the alternate source, issue the **set refresh-from** command at the [edit system scripts commit file *filename*] hierarchy level, specifying the location of the remote file:

```
[edit system scripts commit file filename]
user@host# set refresh-from url
```

To update all commit scripts from the alternate source, issue the **set refresh-from** command at the [edit system scripts commit] hierarchy level, specifying the location of the remote directory that houses the scripts:

```
[edit system scripts commit]
user@host# set refresh-from url
```

At both hierarchy levels:

url—URL of the remote commit script or directory. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

When you issue the **set refresh-from** command, the router attempts to connect to the machine that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local commit scripts directory. The updated commit script is executed when you next issue the **commit** command. If no master source for a script is defined, it is not updated and a warning is issued.

If the router has dual Routing Engines and you want to update a script on both Routing Engines, you must issue the **set refresh-from** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh-from** statement to update scripts on both Routing Engines.

Executing Large Commit Scripts

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

Directly accessing the configuration data and processing non-XML converted data are processor-intensive 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, include the `direct-access` statement at the `[edit system scripts commit]` hierarchy level.

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

Converting a Commit 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 “SLAX Overview” on page 27.

To convert an XSLT script to SLAX, issue the `request system scripts convert xslt-to-slax` source *source-directory/source-filename* destination *destination-directory/<destination-filename>* operational mode command. 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/script1.slax
```

The `script1.xsl` file remains unchanged in the `/var/db/scripts/commit` directory, and a new script called `script1.slax` is added to the `/var/db/scripts/commit` directory. If you do not specify a filename for the SLAX file, it is named `SLAX-Conversion-Temp.xxxxx` where `xxxxx` is a randomly generated series of characters.

To convert a script from SLAX to XSLT, see “Converting a Commit Script from SLAX to XSLT” on page 175.

Converting a Commit Script from SLAX to XSLT

To convert a SLAX script to XSLT, issue the `request system scripts convert slax-to-xslt` source *source-directory/source-filename* destination *destination-directory/<destination-filename>* operational mode command. 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/script1.xsl
```

The `script1.slax` file remains unchanged in the `/var/db/scripts/commit` directory, and a new script called `script1.xsl` is added to the `/var/db/scripts/commit` directory. If

you do not specify a filename for the XSLT file, it is named `SLAX-Conversion-Temp.xxxx` where `xxxx` is a randomly generated series of characters.

To convert a script from XSLT to SLAX, see “Converting a Commit Script from XSLT to SLAX” on page 175.

Displaying Commit Script Output

Table 11 on page 176 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 177.

Table 11: Commit Script Configuration and Operational Mode Commands

| Task | Command |
|--|--|
| Configuration Mode Commands | |
| Display errors and warnings generated by commit scripts. | <code>commit</code> or <code>commit check</code> |
| Display detailed information. | <code>commit display detail</code> |
| Display the underlying Extensible Markup Language (XML) data. | <code>commit display xml</code> |
| Display the postinheritance contents of the configuration database. This view includes transient changes, but does not include changes made in configuration groups. | <code>show display commit-scripts</code> |
| Display the postinheritance contents of the configuration database. This view excludes transient changes. | <code>show display commit-scripts no-transients</code> |
| Display the postinheritance configuration in XML format. | <code>show display commit-scripts view</code> |
| Viewing the configuration in XML format can be helpful when you are writing XML Path Language (XPath) expressions and configuration element tags. | |
| Display the postinheritance configuration in XML format, but exclude transient changes. | <code>show display commit-scripts view display commit-scripts no-transients</code> |
| Display all configuration groups data, including script-generated changes to the groups. | <code>show groups display commit-scripts</code> |
| Display a particular configuration group, including script-generated changes to the group. | <code>show groups group-name display commit-scripts</code> |

Table 11: Commit Script Configuration and Operational Mode Commands *(continued)*

| Task | Command |
|--|---|
| Operational Mode Commands | |
| Display logging data associated with all commit script processing. | <code>show log cscript.log</code> |
| Display processing for only the most recent commit operation. | <code>show log cscript.log last</code> |
| Display processing for script errors. | <code>show log cscript.log match error</code> |
| Display processing for a particular script. | <code>show log cscript.log match script-name</code> |

Tracing Commit Script Processing

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

The default operation of commit script tracing is to log important events in a file called `cscript.log` located in the `/var/log` directory. When the file `cscript.log` reaches 128 kilobytes (KB), it is renamed with a number 0 through 9 (in ascending order) appended to the end of the file and then compressed. For example, the log file is saved as `cscript.log.0.gz`, then `cscript.log.1.gz` until there are 10 trace files. Then the oldest trace file (`cscript.log.9.gz`) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

This section discusses the following topics:

- Minimum Configuration for Tracing for Commit Script Operations on page 177
- Configuring Tracing of Commit Scripts on page 179

Minimum Configuration for Tracing for Commit Script Operations

If no commit script trace options are configured, the simplest way to view the trace output of a commit script is to configure the `output` trace flag and issue the `show log cscript.log | last` command. To do this, perform the following steps:

1. If you have not done so already, enable a commit script by including the `file` statement at the `[edit system scripts commit]` hierarchy level:

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

2. Enable trace options by including the `traceoptions` flag `output` statement at the `[edit system scripts commit]` hierarchy level:

```
[edit system scripts commit]
user@host# set traceoptions flag output
```

3. Issue the commit command:

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

4. Display the resulting trace messages recorded in the file `/var/log/cscript.log`. At the end of the log is the output generated by the commit script you enabled in Step 1. To display the end of the log, issue the `show log cscript.log | last` operational mode command:

```
[edit]
user@host# run show log cscript.log | last
```

Table 12 on page 178 summarizes useful filtering commands that display selected portions of the `cscript.log` file.

Table 12: Commit Script Tracing Operational Mode Commands

| Task | Command |
|--|---|
| Display logging data associated with all script processing. | <code>show log cscript.log</code> |
| Display script processing for only the most recent commit operation. | <code>show log cscript.log last</code> |
| Display processing for script errors. | <code>show log cscript.log match error</code> |
| Display script processing for a particular script. | <code>show log cscript.log match script-name</code> |

Example: Minimum Configuration for Enabling Traceoptions for Commit Scripts

Display the trace output for the commit script file `source-route.xsl`:

```
[edit]
system {
  scripts {
    commit {
      file source-route.xsl;
      traceoptions flag output;
    }
  }
}

[edit]
user@host# commit
[edit]
user@host# run show log cscript.log | last
Jun 20 10:21:24 summary: changes 0, transients 0 (allowed), syslog 0
Jun 20 10:24:15 commit script processing begins
Jun 20 10:24:15 reading commit script configuration
```

```

Jun 20 10:24:15 testing commit script configuration
Jun 20 10:24:15 opening commit script '/var/db/scripts/commit/source-route.xml'
Jun 20 10:24:15 script file '/var/db/scripts/commit/source-route.xml': size=699;
md5 = d947972b429d17ce97fe987d94add6fd
Jun 20 10:24:15 reading commit script 'source-route.xml'
Jun 20 10:24:15 running commit script 'source-route.xml'
Jun 20 10:24:15 processing commit script 'source-route.xml'
Jun 20 10:24:15 results of 'source-route.xml'
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.xml
Jun 20 10:24:15 saving commit script changes
Jun 20 10:24:15 summary: changes 0, transients 0 (allowed), syslog 0

```

Configuring Tracing of Commit Scripts

You cannot change the directory (`/var/log`) to which trace files are written. However, you can customize other trace file settings by including the following statements at the `[edit system scripts commit traceoptions]` hierarchy level:

```

[edit system scripts commit traceoptions]
file <filename> <files number> <size size> <world-readable | no-world-readable>;
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
no-remote-trace;

```

These statements are described in the following sections:

- Configuring the Commit Script Log Filename on page 179
- Configuring the Number and Size of Commit Script Log Files on page 180
- Configuring Access to Commit Script Log Files on page 180
- Configuring the Commit Script Trace Operations on page 180

Configuring the Commit Script Log Filename

By default, the name of the file that records trace output is `cscript.log`. You can specify a different name by including the `file` statement at the `[edit system scripts commit traceoptions]` hierarchy level:

```

[edit system scripts commit traceoptions]
file filename;

```

Configuring the Number and Size of Commit Script Log Files

By default, when the trace file reaches 128 KB in size, it is renamed and compressed to *filename.0.gz*, then *filename.1.gz*, and so on, until there are 10 trace files. Then the oldest trace file (*filename.9.gz*) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the [edit system scripts commit traceoptions file <filename>] hierarchy level:

```
[edit system scripts commit traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 640 KB and the maximum number of files to 20. When the file that receives the output of the tracing operation (*filename*) reaches 640 KB, it is renamed and compressed to *filename.0.gz*, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename.0.gz* is renamed *filename.1.gz* and *filename* is renamed and compressed to *filename.0.gz*. This process repeats until there are 20 trace files. Then the oldest file (*filename.19.gz*) is overwritten.

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

If you set either a maximum file size or a maximum number of trace files, you also must specify the other parameter and a filename.

Configuring Access to Commit Script Log Files

By default, access to the commit script log file is restricted to the owner. You can manually configure access by including the *world-readable* or *no-world-readable* statement at the [edit system scripts commit traceoptions file <filename>] hierarchy level.

```
[edit system scripts commit traceoptions file <filename>]
(world-readable | no-world-readable);
```

The *no-world-readable* statement restricts commit script log access to the owner. The *world-readable* statement enables unrestricted access to the commit script log file.

Configuring the Commit Script Trace Operations

By default, only important events are logged. You can configure the trace operations to be logged by including the following statements at the [edit system scripts commit traceoptions] hierarchy level:

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

Table 13 on page 181 describes the meaning of the commit script tracing flags.

Table 13: 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 14 on page 181 describes some common problems that might occur.

Table 14: 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/ <i>filename</i> : No such file or directory | Make sure the file <i>filename</i> 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: <i>filename</i> 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: <ul style="list-style-type: none"> ■ 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 14: Troubleshooting Commit Scripts (*continued*)

| Problem | Solution |
|---|---|
| <p>An expected action does not occur.</p> <p>For example, a warning message does not appear even though the configuration contains the problem that is supposed to evoke the warning message.</p> | <ol style="list-style-type: none"> 1. Make sure you have enabled the script. Scripts are ignored if they are not enabled. To enable a script, include the <code>file filename</code> statement at the <code>[edit system scripts commit]</code> hierarchy level. 2. Make sure you have included the required boilerplate in your script. For more information, see “Required Boilerplate for Commit Scripts” on page 115. 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 <i>JUNOS XML API Configuration Reference</i> or by issuing the <code>show configuration display xml</code> 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 <code><xsl:for-each></code> instruction contains an XPath expression, which changes the context node. So the nested <code><xsl:if></code> instruction uses an XPath expression that is relative to the <code>interfaces/interface[starts-with(name, 't1-')]</code> XPath expression. <pre> <xsl:for-each select="interfaces/interface[starts-with(name, 't1-')]"> <xsl:if test="not(description)"> </pre> |

Chapter 14

Commit Script Examples

This chapter includes the following examples:

- Example: Requiring and Restricting Configuration Statements on page 183
- Example: Requiring Internal Clocking on T1 Interfaces on page 187
- Example: Imposing a Minimum MTU Setting on page 189
- Example: Limiting the Number of E1 Interfaces on page 191
- Example: Limiting the Number of ATM Virtual Circuits on page 200
- Example: Controlling IS-IS and MPLS Interfaces on page 203
- Example: Adding T1 Interfaces to a RIP Group on page 206
- Example: Configuring a Default Encapsulation Type on page 209
- Example: Controlling LDP Configuration on page 212
- Example: Adding a Final then accept Term to a Firewall on page 216
- Example: Configuring an Interior Gateway Protocol on an Interface on page 220
- Example: Creating a Complex Configuration Based on a Simple Interface Configuration on page 224
- Example: Configuring Administrative Groups for LSPs on page 230
- Example: Configuring Dual Routing Engines on page 234
- Example: Preventing Import of the Full Routing Table on page 238
- Example: Automatically Configuring Logical Interfaces and IP Addresses on page 241
- Example: Prepending a Global Policy on page 247
- Example: Assigning a Classifier on page 252
- Example: Loading a Base Configuration on page 255

Example: 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 the ex-no-nukes Script

To test the ex-no-nukes script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Requiring and Restricting Configuration Statements” on page 183 into a text file, name the file `ex-no-nukes.xml` or `ex-no-nukes.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-no-nukes.slax`.

```
system {
  scripts {
    commit {
      file ex-no-nukes.xml;
    }
  }
}
interfaces {
  fxp0 {
    disable;
    unit 0 {
      family inet {
        address 10.0.0.1/24;
      }
    }
  }
}
```

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

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

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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
```

Example: 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 the ex-clocking-error Script

To test the `ex-clocking-error` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Requiring Internal Clocking on T1 Interfaces” on page 187 into a text file, name the file `ex-clocking-error.xml` or `ex-clocking-error.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-clocking-error.slax`.

```

system {
  scripts {
    commit {
      file ex-clocking-error.xml;
    }
  }
}
interfaces {
  t1-0/0/0 {
    clocking external;
  }
  t1-0/0/1 {
    unit 0;
  }
}

```

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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

Example: Imposing a Minimum MTU Setting

The maximum transmission unit (MTU) is the greatest amount of data or packet size (in bytes) that can be transferred in one physical frame on a network.

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 < $min-mtu]">
      <xnm:error>
        <xsl:call-template name="jcs:edit-path"/>
        <xsl:call-template name="jcs:statement">
          <xsl:with-param name="dot" select="mtu"/>
        </xsl:call-template>
        <message>
          <xsl:text>SONET interfaces must have a minimum MTU of </xsl:text>
          <xsl:value-of select="$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 the ex-so-mtu Script

To test the ex-so-mtu script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Imposing a Minimum MTU Setting” on page 189 into a text file, name the file `ex-so-mtu.xml` or `ex-so-mtu.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-so-mtu.slax`.

```

system {
  scripts {
    commit {
      file ex-so-mtu.xml;
    }
  }
}
interfaces {
  so-1/2/2 {
    mtu 2048;
  }
  so-1/2/3 {
    mtu 576;
  }
}

```

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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
```

Example: 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.xml"/>

  <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.xml";

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 the ex-16-e1-limit Script

To test the ex-16-e1-limit script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Limiting the Number of E1 Interfaces” on page 191 into a text file, name the file **ex-16-e1-limit.xml** or **ex-16-e1-limit.slax** as appropriate, and copy it to the **/var/db/scripts/commit** directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **ex-16-e1-limit.slax**.

```

system {
  scripts {
    commit {
      file ex-16-e1-limit.xml;
    }
  }
}
interfaces {
  cau4-0/1/0 {
    partition 1 interface-type ce1;
    partition 2-18 interface-type e1;
  }
  cstm1-0/1/0 {
    no-partition interface-type cau4;
  }
  ce1-0/1/0:1 {
    clocking internal;
  }
}

```

```

    e1-options {
        framing g704;
    }
    partition 1 timeslots 1-4 interface-type ds;
}
ds-0/1/0:1:1 {
    no-keepalives;
    dce;
    encapsulation frame-relay;
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.0/31;
        }
    }
}
e1-0/1/0:2 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.2/31;
        }
    }
}
e1-0/1/0:3 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {

```

```

        address 10.0.0.4/31;
    }
}
e1-0/1/0:4 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.6/31;
        }
    }
}
e1-0/1/0:5 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.8/31;
        }
    }
}
e1-0/1/0:6 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
}

```

```

    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.10/31;
        }
    }
}
e1-0/1/0:7 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.12/31;
        }
    }
}
e1-0/1/0:8 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.14/31;
        }
    }
}
e1-0/1/0:9 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
}

```

```

    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.16/31;
        }
    }
}
e1-0/1/0:10 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.18/31;
        }
    }
}
e1-0/1/0:11 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.20/31;
        }
    }
}
e1-0/1/0:12 {
    no-keepalives;
    per-unit-scheduler;
    dce;

```

```

clocking internal;
encapsulation frame-relay;
e1-options {
    framing g704;
}
lmi {
    lmi-type ansi;
}
unit 100 {
    point-to-point;
    dlci 100;
    family inet {
        address 10.0.0.22/31;
    }
}
}
e1-0/1/0:13 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.24/31;
        }
    }
}
e1-0/1/0:14 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.26/31;
        }
    }
}
}

```

```

e1-0/1/0:15 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.28/31;
        }
    }
}
e1-0/1/0:16 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.30/31;
        }
    }
}
e1-0/1/0:17 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {

```



```

        address 10.0.0.32/31;
    }
}
e1-0/1/0:18 {
    no-keepalives;
    per-unit-scheduler;
    dce;
    clocking internal;
    encapsulation frame-relay;
    e1-options {
        framing g704;
    }
    lmi {
        lmi-type ansi;
    }
    unit 100 {
        point-to-point;
        dlci 100;
        family inet {
            address 10.0.0.34/31;
        }
    }
}
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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

```

Example: 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.xml"/>

  <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 > $limit">
        <xnm:error>
          <edit-path>[edit interfaces]</edit-path>
          <statement><xsl:value-of select="name" /></statement>
          <message>
            <xsl:text>ATM VC limit exceeded; </xsl:text>
            <xsl:value-of select="$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.xml";

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 the *ex-atm-vc-limit* Script

To test the *ex-atm-vc-limit* script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Limiting the Number of ATM Virtual Circuits” on page 200 into a text file, name the file *ex-atm-vc-limit.xml* or *ex-atm-vc-limit.slax* as appropriate, and copy it to the */var/db/scripts/commit* directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to *ex-atm-vc-limit.slax*.

```

system {
    scripts {
        commit {
            file ex-atm-vc-limit.xml;
        }
    }
}
interfaces {
    at-1/2/3 {
        unit 15 {
            family inet {
                address 10.12.13.15/20;
            }
        }
        unit 16 {
            family inet {
                address 10.12.13.16/20;
            }
        }
        unit 17 {
            family inet {
                address 10.12.13.17/20;
            }
        }
        unit 18 {
            family inet {
                address 10.12.13.18/20;
            }
        }
        unit 19 {

```

```

        family inet {
            address 10.12.13.19/20;
        }
    }
    unit 20 {
        family inet {
            address 10.12.13.20/20;
        }
    }
    unit 21 {
        family inet {
            address 10.12.13.21/20;
        }
    }
    unit 22 {
        family inet {
            address 10.12.13.22/20;
        }
    }
    unit 23 {
        family inet {
            address 10.12.13.23/20;
        }
    }
    unit 24 {
        family inet {
            address 10.12.13.24/20;
        }
    }
    unit 25 {
        family inet {
            address 10.12.13.25/20;
        }
    }
    unit 26 {
        family inet {
            address 10.12.13.26/20;
        }
    }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
- b. Press Enter.
- c. Press Ctrl + d.

4. 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
```

Example: Controlling IS-IS and MPLS Interfaces

If you want to enable MPLS on an interface, you must make changes at both the [edit interfaces] and [edit protocols mpls] hierarchy levels. This example shows 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 (loX), 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.xml"/>

  <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:call-template>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
```

```

        </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 the ex-iso Script

To test the `ex-iso` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Controlling IS-IS and MPLS Interfaces” on page 203 into a text file, name the file `ex-iso.xml` or `ex-iso.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to `ex-iso.slax`.

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

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

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

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
- b. Press Enter.
- c. Press Ctrl + d.

4. 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;
  }
}
```

Example: Adding T1 Interfaces to a RIP Group

If you want to enable RIP on an interface, you must make changes at both the `[edit interfaces]` and `[edit protocols rip]` hierarchy levels. This example shows 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.xml"/>
```



```

<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.xml";

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 the *ex-rip-t1* Script

To test the *ex-rip-t1* script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Adding T1 Interfaces to a RIP Group” on page 206 into a text file, name the file *ex-rip-t1.xsl* or *ex-rip-t1.slax* as appropriate, and copy it to the */var/db/scripts/commit* directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to *ex-rip-t1.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. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your router configuration:

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

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. Issue the **commit** command and then the **show protocols rip group test** command. The following output appears:

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

Example: Configuring 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.

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

XSLT Syntax

```

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

  <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.xml";

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 the ex-so-encap Script

To test the ex-so-encap script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Configuring a Default Encapsulation Type” on page 209 into a text file, name the file **ex-so-encap.xml** or **ex-so-encap.slax** as appropriate, and copy it to the **/var/db/scripts/commit** directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to **ex-so-encap.slax**.

```

system {
  scripts {
    commit {
      allow-transients;
      file ex-so-encap.xml;
    }
  }
}

```

```

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

```

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

```

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

```

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

```

[edit]
user@host# show interfaces
so-1/2/2 {
  sonet-options {

```

```

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

```

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

```

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

```

Example: Controlling LDP Configuration

If you want to enable LDP on an interface, you must configure the interface at both the `[edit protocols routing-protocol-name]` and `[edit protocols ldp]` hierarchy levels. This example shows 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:

```
[edit]
protocols {
  isis {
    interface so-0/1/2.0 {
      apply-macro no-ldp;
    }
  }
}
```

If the `apply-macro no-ldp` statement is included, the warning is not 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.xml"/>

  <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.xml";

apply-macro no-ldp;match configuration {
    var $ldp = protocols/ldp;
    var $isis = protocols/isis;
    var $ospf = protocols/ospf;
    if ($ldp) {
        for-each ($isis/interface/name | $ospf/area/interface/name) {
            var $ifname = .;
            if (not(../apply-macro[name = 'no-ldp']) and not($ldp/interface[name =
                $ifname])) {
                <xnm:warning> {
                    call jcs:edit-path();
                    call jcs:statement();
                    <message> "ldp not enabled for this interface";
                }
            }
        }
    }
    for-each (protocols/ldp/interface/name) {
        var $ifname = .;
        if (not(apply-macro[name = 'no-igp']) and not($isis/interface[name =
            $ifname]) and not($ospf/area/interface[name = $ifname])) {
            <xnm:warning> {
                call jcs:edit-path();
                call jcs:statement();
                <message> {
                    expr "ldp-enabled interface does not have ";
                    expr "an IGP configured";
                }
            }
        }
    }
}

```


Testing the ex-ldp Script

To test the `ex-ldp` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Controlling LDP Configuration” on page 212 into a text file, name the file `ex-ldp.xml` or `ex-ldp.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-ldp.slax`.

```
system {
  scripts {
    commit {
      file ex-ldp.xml;
    }
  }
}
protocols {
  isis {
    interface so-1/2/2.0 {
      apply-macro no-ldp;
    }
    interface so-1/2/3.0;
  }
  ospf {
    area 10.4.0.0 {
      interface ge-3/2/1.0;
      interface ge-2/2/1.0;
    }
  }
  ldp {
    interface ge-1/2/1.0;
    interface ge-2/2/1.0;
  }
}
```

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

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

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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
```

Example: 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.xml"/>

  <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.xml";

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 the *ex-add-accept* Script

To test the *ex-add-accept* script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Adding a Final then accept Term to a Firewall” on page 216 into a text file, name the file *ex-add-accept.xml* or *ex-add-accept.slax* as appropriate, and copy it to the */var/db/scripts/commit* directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to *ex-add-accept.slax*.

```

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

```

```

    }
  }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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 */
    }
}
}

```

Example: Configuring an Interior Gateway Protocol on an Interface

When you add a new interface to an OSPF or IS-IS domain, you must configure the interface at multiple hierarchy levels, including [edit interfaces] and [edit protocols]. This example uses a 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 (mgd) process inspects the configuration, looking for `apply-macro` statements. For each `apply-macro ifclass` statement included at the [edit interfaces *interface-name* unit *logical-unit-number*] hierarchy level, the script tests whether the `role` parameter is defined as `cpe`. If so, the script checks the `igp` parameter.

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

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

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.xml"/>

  <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.xml";

```

```

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 the ex-if-class Script

To test the ex-if-class script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Configuring an Interior Gateway Protocol on an Interface” on page 220 into a text file, name the file `ex-if-class.xml` or `ex-if-class.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-if-class.slax`.

```

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

```


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

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

- a. At the prompt, paste the contents of the clipboard 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;
    }
  }
}
```

Example: 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 (mgd) process inspects the configuration, looking for `apply-macro params` statements included at the `[edit interfaces interface-name]` hierarchy level.

When the script finds an `apply-macro params` statement, it performs the following actions:

- Applies the `interface-details` configuration group to the interface.
- Includes the value of the `description` parameter at the `[edit interfaces interface-name description]` hierarchy level.
- Includes the value of the `encapsulation` parameter at the `[edit interfaces interface-name encapsulation]` hierarchy level. If the `encapsulation` parameter is not included in the `apply-macro params` statement, the script sets the encapsulation to `cisco-hdlc` as 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($encapsulation) > 0">
                  <xsl:value-of select="$encapsulation"/>
                </xsl:when>
                <xsl:otherwise>cisco-hdlc</xsl:otherwise>
              </xsl:choose>
            </encapsulation>
            <unit>
              <name>0</name>
              <xsl:if test="string-length($inet-address) > 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 the ex-if-params Script

To test the `ex-if-params` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Creating a Complex Configuration Based on a Simple Interface Configuration” on page 224 into a text file, name the file `ex-if-params.xml` or `ex-if-params.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-if-params.slax`.

```
system {
  scripts {
    commit {
      allow-transients;
      file ex-if-params.xml;
    }
  }
}
groups {
```

```

interface-details {
  interfaces {
    <so-*/*/*> {
      clocking internal;
    }
  }
}
interfaces {
  so-1/2/3 {
    apply-macro params {
      description "Link to Hoverville";
      encapsulation ppp;
      inet-address 10.1.2.3/28;
      isis-level-1 enable;
      isis-level-1-metric 50;
      isis-level-2-metric 85;
    }
  }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard 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;
}

```

Example: Configuring Administrative Groups for LSPs

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

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 the ex-lsp-admin Script” on page 232.

For a line-by-line explanation of this script, see “Example: Creating Custom Configuration Syntax with Macros” on page 161.

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 the *ex-lsp-admin* Script

To test the *ex-lsp-admin* script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Configuring Administrative Groups for LSPs” on page 230 into a text file, name the file *ex-lsp-admin.xml* or *ex-lsp-admin.slax* as appropriate, and copy it to the */var/db/scripts/commit* directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the *[edit system scripts commit file]* hierarchy level to *ex-lsp-admin.slax*.

```

system {
  scripts {
    commit {
      allow-transients;
      file ex-lsp-admin.xml;
    }
  }
}
protocols {
  mpls {
    apply-macro blue-type-lsp {
      10.1.1.1;
      10.2.2.2;
    }
  }
}

```

```

        10.3.3.3;
        10.4.4.4;
        color blue;
        group-value 0;
    }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard 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;
}

```

Example: Configuring Dual Routing Engines

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.xml 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.xml"/>

  <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.xml 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.xml"/>

  <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>
        </transient-change>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

```

        </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.xml 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.xml";

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.xml 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.xml";

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 the ex-dual-re and ex-dual-re2 Scripts

To test the `ex-dual-re` and `ex-dual-re2` scripts, perform the following steps:

1. Copy the XSLT or SLAX scripts from “Example: Configuring Dual Routing Engines” on page 234 into two text files, name the files `ex-dual-re.xml` and `ex-dual-re2.xml` or `ex-dual-re.slax` and `ex-dual-re2.slax` as appropriate, and copy them to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filenames at the [edit system scripts commit file] hierarchy level to `ex-dual-re.slax` and `ex-dual-re2.slax`.

```
groups {
  re0 {
    interfaces {
      fxp0 {
        unit 0 {
          family inet {
            address 10.0.0.1/24;
          }
        }
      }
    }
  }
}
apply-groups re0;
```

```

system {
  host-name router1;
  scripts {
    commit {
      file ex-dual-re.xml;
      file ex-dual-re2.xml;
    }
  }
}
interfaces {
  fe-0/0/0 {
    unit 0 {
      family inet {
        address 192.168.220.1/30;
      }
    }
  }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. Issue the **commit** command. The following output appears. After the commit operation completes, the router hostname is changed to **router1-RE0**.

```

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

```

Example: 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 the ex-import Script

To test the **ex-import** script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Preventing Import of the Full Routing Table” on page 238 into a text file, name the file **ex-import.xml** or **ex-import.slax** as appropriate, and copy it to the **/var/db/scripts/commit** directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **ex-import.slax**.

```

system {
  scripts {
    commit {
      file ex-import.xml;
    }
  }
}
protocols {
  ospf {
    import bad-news;
  }
}
policy-options {
  policy-statement bad-news {
    then accept;
  }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
 - b. Press Enter.
 - c. Press Ctrl + d.
4. 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
```

Example: Automatically Configuring Logical Interfaces and IP Addresses

Every interface you configure requires at least one logical unit and one IP address. Asynchronous Transfer Mode (ATM) interfaces also require a virtual circuit identifier (VCI) for each logical interface. If you need to configure multiple logical units on an interface, you can use a commit script 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.xml"/>

  <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($max) > 0">
            <xsl:value-of select="$max"/>
          </xsl:when>
          <xsl:otherwise>0</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:variable name="real-unit">
        <xsl:choose>
          <xsl:when test="string-length($unit) > 0">
            <xsl:value-of select="$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:for-each>
  </xsl:template>
</xsl:stylesheet>
```

```

        </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 > $unit">
      <xsl:call-template name="emit-interface">
        <xsl:with-param name="address">
          <xsl:call-template name="next-address">
            <xsl:with-param name="address" select="$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) > 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($arg-prefix) > 0">
        <xsl:value-of select="$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="$a4 < 255">
        <xsl:value-of select="$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 < 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 the ex-atm-logical Script

To test the ex-atm-logical script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Automatically Configuring Logical Interfaces and IP Addresses” on page 241 into a text file, name the file `ex-atm-logical.xml` or `ex-atm-logical.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-atm-logical.slax`.

```
system {
  scripts {
    commit {
      allow-transients;
      file ex-atm-logical.xml;
    }
  }
}
interfaces {
  apply-macro at-1/2/3 {
    address 10.12.13.14/20;
    max 200;
    unit 32;
  }
  at-1/2/3 {
    atm-options {
      pic-type atm2;
      vpi 0;
    }
  }
}
```

3. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

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

- a. At the prompt, paste the contents of the clipboard 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;
    }
}
```

Example: 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 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 the ex-bgp-global-import Script

To test the ex-bgp-global-import script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Prepending a Global Policy” on page 247 into a text file, name the file `ex-bgp-global-import.xsl` or `ex-bgp-global-import.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the [edit system scripts commit file] hierarchy level to `ex-bgp-global-import.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. In configuration mode, issue the **load merge terminal** command to merge the stanzas into your router configuration:

```

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

```

- a. At the prompt, paste the contents of the clipboard 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:
- ```
[edit]
user@host# show protocols
bgp {
 group fish {
 neighbor 192.168.16.4 {
 import [blue green];
 peer-as 65401;
 }
 neighbor 192.168.16.6 {
 peer-as 65402;
 }
 }
}
```
- 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:
- ```
[edit]
user@host# show protocols | display commit-scripts
bgp {
  import bgp_global_import;
  group fish {
    neighbor 192.168.16.4 {
      import [ bgp_global_import blue green ];
      peer-as 65401;
    }
    neighbor 192.168.16.6 {
      peer-as 65402;
    }
  }
}
```
- 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:
- ```
[edit]
user@host# set protocols bgp group fish neighbor 192.168.16.6 import green
[edit]
user@host# show protocols | display commit-scripts
bgp {
 import bgp_global_import;
 group fish {
 neighbor 192.168.16.4 {
 import [bgp_global_import blue green];
 peer-as 65401;
 }
 neighbor 192.168.16.6 {
 import [bgp_global_import blue green];
 peer-as 65402;
 }
 }
}
```

```
}
```

## Example: Assigning a Classifier

In JUNOS Software 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.xml"/>

 <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.xml";

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 the ex-classifier Script

To test the ex-classifier script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Assigning a Classifier” on page 252 into a text file, name the file `ex-classifier.xml` or `ex-classifier.slax` as appropriate, and copy it to the `/var/db/scripts/commit` directory on the router.
2. Select the following configuration stanzas, and press Ctrl + c to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the `[edit system scripts commit file]` hierarchy level to `ex-classifier.slax`.

```

system {
 scripts {
 commit {
 file ex-classifier.xml;
 }
 }
}
interfaces {
 fe-0/0/0 {
 unit 0 {
 family inet {
 address 10.168.16.2/24;
 }
 }
 }
}

```

```

}
class-of-service {
 forwarding-classes {
 queue 3 fc-q3;
 }
 classifiers {
 inet-precedence fc-q3 {
 forwarding-class fc-q3 {
 loss-priority low code-points 010;
 }
 }
 }
}

```

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

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
  - b. Press Enter.
  - c. Press Ctrl + d.
4. Issue the **commit** command. The following output appears:

```

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

```

The output from the **show class-of-service** configuration mode command now shows that the **fe-0/0/0.0** interface has been assigned the **fc-q3** classifier:

```

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

```



```
}
}
```

## Example: 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="$bkup-rtr"/></address>
 </backup-router>
 <time-zone>America/Los_Angeles</time-zone>
 <authentication-order>radius</authentication-order>
 <authentication-order>password</authentication-order>
 <root-authentication>
 <encrypted-password>
 1Q3CG88jZ$.qhPUZaHdaIMWF2CvxKTeO
 </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/AHkmTQFCA0B1cIKWL7sgaRh-bs4GU
 </secret>
 </radius-server>
 <radius-server>
 <name>192.168.4.240</name>
 <secret>
 9TQ/t1lcSrKA0IRheK8X7VYgaZDm5zNdiqmTn6
 </secret>
 </radius-server>
 </system>
 </transient-change>
 </xsl:otherwise>
</xsl:choose>

```

```

</radius-server>
<login>
 <class>
 <permissions>all</permissions>
 </class>
 <user>
 <name>johnny</name>
 <uid>928</uid>
 <class>superuser</class>
 <authentication>
 <encrypted-password>
 1kPU..$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>
 </file>
</syslog>

```

```

 <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>
 </community>

```

```

 <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.xml";

```

```

var $macro-name = 'config-system.xml';
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> $bkup-rtr;
 }
 <time-zone> "America/Los_Angeles";
 <authentication-order> "radius";
 <authentication-order> "password";
 <root-authentication> {
 <encrypted-password>
 "1Q3CG88jZ$.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> "9TQ/t1lcSrKAt0IRheK8X7VYgaZDm5zNdiqmTn6";
 }
 <login> {
 <class> {
 <permissions> "all";
 }
 <user> {
 <name> "johnny";
 <uid> "928";
 <class> "superuser";
 <authentication> {
 <encrypted-password> "1kPU..$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 the config-system Script

To test the `config-system` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Loading a Base Configuration” on page 255 into a text file, name the file **config-system.xml** or **config-system.slax** as appropriate, and copy it to the **/var/db/scripts/commit** directory on the router.
2. Select the following configuration stanzas, and press **Ctrl + c** to copy them to the clipboard. If you are using the SLAX version of the script, change the filename at the **[edit system scripts commit file]** hierarchy level to **config-system.slax**.

```
system {
```

```

scripts {
 commit {
 allow-transients;
 file config-system.xml;
 }
}
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. In configuration mode, issue the `load merge terminal` command to merge the stanzas into your router configuration:

```

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

```

- a. At the prompt, paste the contents of the clipboard using the mouse and the paste icon.
  - b. Press Enter.
  - c. Press Ctrl + d.
4. Issue the `commit` command.

```

[edit]
user@host# commit

```

After committing the configuration, issue the `show | display commit-scripts` configuration mode command to view the router base configuration:

```

user@host# show | display commit-scripts
...

```

## Chapter 15

# **Summary of JUNOS XML and XSLT Tag Elements Used in Commit Scripts**

This chapter lists the JUNOS XML tag elements used to generate custom warning, error, and system log (syslog) messages, and the XSLT tag elements used to make custom permanent or transient changes. The tag elements are in alphabetical order.

**<change> (XSLT)**

---

**Usage**

```
<change 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/>** element'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/>** element'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** See “Overview of Generating Persistent or Transient Configuration Changes” on page 137 and “Overview of Creating Custom Configuration Syntax with Macros” on page 153.

**Related Topics** ■ <transient-change> (XSLT) on page 272

## <syslog> (JUNOS XML)

---

**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** See “Generating a Custom Warning, Error, or System Log Message” on page 124.

**<transient-change> (XSLT)**

---

**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/>` element's `url` attribute to the pathname of a file that resides on the routing platform and contains the JUNOS XML-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**    See “Overview of Generating Persistent or Transient Configuration Changes” on page 137 and “Overview of Creating Custom Configuration Syntax with Macros” on page 153.

**Related Topics**    ■    `<change>` (XSLT) on page 270

**<xnm:error> (JUNOS XML)**

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Usage</b>               | <pre> &lt;xnm:error xmlns="namespace-URL" xmlns:xnm="namespace-URL"&gt;   &lt;parse/&gt;   &lt;source-daemon&gt;module-name&lt;/source-daemon&gt;   &lt;filename&gt;filename&lt;/filename&gt;   &lt;line-number&gt;line-number&lt;/line-number&gt;   &lt;column&gt;column-number&lt;/column&gt;   &lt;token&gt;input-token-id&lt;/token&gt;   &lt;edit-path&gt;edit-path-name&lt;/edit-path&gt;   &lt;statement&gt;statement-string&lt;/statement&gt;   &lt;message&gt;error-string&lt;/message&gt;   &lt;re-name&gt;re-name-string&lt;/re-name&gt;   &lt;database-status-information&gt;user&lt;/database-status-information&gt;   &lt;reason&gt;reason-string&lt;/reason&gt; &lt;/xnm:error&gt; </pre>                                                                                                                                                                                                                                                                                                                                           |
| <b>Release Information</b> | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b>         | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Attributes</b>          | <p><b>xmlns</b>—Names the XML namespace for the contents of the tag element. The value is a URL of the form <code>http://xml.juniper.net/xnm/version/xnm</code>, where <i>version</i> is a string such as <code>1.1</code>.</p> <p><b>xmlns:xnm</b>—Names the XML namespace for child tag elements that have the <b>xnm:</b> prefix on their names. The value is a URL of the form <code>http://xml.juniper.net/xnm/version/xnm</code>, where <i>version</i> is a string such as <code>1.1</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Contents</b>            | <p><b>&lt;column&gt;</b>—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 <b>&lt;line-number&gt;</b> tag element in the configuration file that was being loaded (which is named in the <b>&lt;filename&gt;</b> tag element).</p> <p><b>&lt;database-status-information&gt;</b>—Provides information about the users currently editing the configuration.</p> <p><b>&lt;edit-path&gt;</b>—Specifies the command-line interface (CLI) configuration mode edit path in effect when the error occurred (provided only during loading of a configuration file).</p> <p><b>&lt;filename&gt;</b>—Names the configuration file that was being loaded.</p> <p><b>&lt;line-number&gt;</b>—Specifies the line number where the error occurred in the configuration file that was being loaded, which is named by the <b>&lt;filename&gt;</b> tag element.</p> <p><b>&lt;message&gt;</b>—Describes the error in a natural-language text string.</p> |

**<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** See “Generating a Custom Warning, Error, or System Log Message” on page 124.

**Related Topics** ■ `<xnm:warning>` (JUNOS XML) on page 275

**<xnm:warning> (JUNOS XML)**

---

**Usage**      `<xnm:warning xmlns="namespace-URL" xmlns:xnm="namespace-URL">`  
                  `<source-daemon>module-name</source-daemon>`  
                  `<filename>filename</filename>`  
                  `<line-number>line-number</line-number>`  
                  `<column>column-number</column>`  
                  `<token>input-token-id</token>`  
                  `<edit-path>edit-path-name</edit-path>`  
                  `<statement>statement-name</statement>`  
                  `<message>error-string</message>`  
                  `<reason>reason-string</reason>`  
                  `</xnm:warning>`

**Release Information**    Statement introduced in JUNOS 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** See “Generating a Custom Warning, Error, or System Log Message” on page 124

**Related Topics** ■ <xnm:error> (JUNOS XML) on page 273.

## Chapter 16

# Summary of Commit Script Configuration Statements

This chapter describes each configuration statement for commit scripts. The statements are organized alphabetically.

### allow-transients

---

|                                 |                                                                                                                                                                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | allow-transients;                                                                                                                                                                                                                                      |
| <b>Hierarchy Level</b>          | [edit system scripts commit]                                                                                                                                                                                                                           |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                             |
| <b>Description</b>              | For JUNOS commit scripts, enable transient configuration changes to be committed.                                                                                                                                                                      |
| <b>Default</b>                  | Transient changes are disabled by default. If you do not include the <b>allow-transients</b> statement, and an enabled script generates transient changes, the command-line interface (CLI) generates an error message and the commit operation fails. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                            |
| <b>Related Topics</b>           | <ul style="list-style-type: none"><li>■ Generating a Persistent or Transient Change on page 141</li><li>■ Creating a Macro to Read the Custom Syntax and Generate Related Configuration Statements on page 159</li></ul>                               |

## 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.

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

**Related Topics**    ■    Overview of Creating Custom Configuration Syntax with Macros on page 153



## commit

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <pre> commit {   allow-transients;   direct-access;   file <i>filename</i> {     optional;     refresh;     refresh-from <i>url</i>;     source <i>url</i>;   }   refresh;   refresh-from <i>url</i>;   traceoptions {     file &lt;<i>filename</i>&gt; &lt;files <i>number</i>&gt; &lt;size <i>size</i>&gt; &lt;world-readable   no-world-readable&gt;;     flag <i>flag</i>;     no-remote-trace;   } } </pre> |
| <b>Hierarchy Level</b>          | [edit system scripts]                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>              | For JUNOS commit scripts, configure the commit-time scripting mechanism.                                                                                                                                                                                                                                                                                                                                         |
| <b>Options</b>                  | The statements are explained separately.                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                                                                                                                                      |
| <b>Related Topics</b>           | ■ Implementing Commit Scripts on page 168                                                                                                                                                                                                                                                                                                                                                                        |

## direct-access

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | direct-access;                                                                                                              |
| <b>Hierarchy Level</b>          | [edit system scripts commit]                                                                                                |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 9.1.                                                                                  |
| <b>Description</b>              | Specify that commit scripts read input configurations directly from the database when inspecting these scripts for errors.  |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Executing Large Commit Scripts on page 174                                                                                |

## file (Commit Scripts)

---

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

The remaining statements are explained separately.

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

**Related Topics**    ■    Controlling Execution of Commit Scripts During Commit Operations on page 168

## 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.

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

**Related Topics**    ■    Controlling Execution of Commit Scripts During Commit Operations on page 168

## refresh (Commit Scripts)

---

|                                 |                                                                                                                                                                                                                                                                                            |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh;                                                                                                                                                                                                                                                                                   |
| <b>Hierarchy Level</b>          | [edit system scripts commit],<br>[edit system scripts commit file <i>filename</i> ]                                                                                                                                                                                                        |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                                                                 |
| <b>Description</b>              | For JUNOS commit scripts, overwrite the local copy of all enabled commit scripts or a single enabled script located in the <code>/var/db/scripts/commit</code> directory with the copy located at the source URL, as specified in the <b>source</b> statement at the same hierarchy level. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh-from (Commit Scripts)</li> <li>■ source (Commit Scripts)</li> <li>■ Updating a Commit Script from the Master Source on page 173</li> </ul>                                                                                                |

## refresh-from (Commit Scripts)

---

|                                 |                                                                                                                                                                                                                                                                      |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh-from <i>url</i> ;                                                                                                                                                                                                                                            |
| <b>Hierarchy Level</b>          | [edit system scripts commit],<br>[edit system scripts commit file <i>filename</i> ]                                                                                                                                                                                  |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                                           |
| <b>Description</b>              | For JUNOS commit scripts, overwrite the local copy of all enabled commit scripts or a single enabled script located in the <code>/var/db/scripts/commit</code> directory with the copy located at a URL other than the URL specified in the <b>source</b> statement. |
| <b>Options</b>                  | <i>url</i> —The source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.                                                                                                                         |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                          |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh (Commit Scripts)</li> <li>■ source (Commit Scripts)</li> <li>■ Updating a Commit Script from an Alternate Location on page 174</li> </ul>                                                                           |

## scripts

---

```

Syntax 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> <world-readable | no-world-readable>;
 flag flag;
 no-remote-trace;
 }
 }
 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;
 no-remote-trace;
 }
 }
 }

```

**Hierarchy Level** [edit system]

**Release Information** Statement introduced in JUNOS Release 7.4.

**Description** For JUNOS commit or op scripts, configure scripting mechanisms.

**Options** The statements are explained separately.

**Required Privilege Level** maintenance—To view this statement in the configuration.

maintenance-control—To add this statement to the configuration.

- Related Topics**
- Implementing Commit Scripts on page 168
  - Implementing Op Scripts on page 300

## source (Commit Scripts)

---

**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.

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

- Related Topics**
- `refresh` (Commit Scripts)
  - `refresh-from` (Commit Scripts)
  - Overview of Updating Commit Scripts from a Remote Source on page 171
  - Configuring the Master Source for a Commit Script on page 173

## traceoptions (Commit and Op Scripts)

---

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>              | <pre> traceoptions {     file &lt;filename&gt; &lt;files number&gt; &lt;size size&gt; &lt;world-readable   no-world-readable&gt;;     flag flag;     no-remote-trace; } </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Hierarchy Level</b>     | [edit system scripts commit],<br>[edit system scripts op]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Release Information</b> | Statement introduced in JUNOS Release 7.4.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Description</b>         | Define tracing operations for commit or op scripts.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Default</b>             | If you do not include this statement, no script-specific tracing operations are performed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Options</b>             | <p><b>filename</b>—Name of the file to receive the output of the tracing operation. All files are placed in the directory <code>/var/log</code>. By default, commit script process tracing output is placed in the file <code>cscript.log</code> and op script process tracing is placed in the file <code>op-script.log</code>. If you include the <b>file</b> statement, you must specify a filename. To retain the default, you can specify <code>cscript.log</code> or <code>op-script.log</code> as the filename.</p> <p><b>files number</b>—(Optional) Maximum number of trace files. When a trace file named <i>trace-file</i> reaches its maximum size, it is renamed and compressed to <i>trace-file.0.gz</i>, then <i>trace-file.1.gz</i>, and so on, until the maximum number of trace files is reached. Then the oldest trace file is overwritten.</p> <p>If you specify a maximum number of files, you also must specify a maximum file size with the <b>size</b> option and a filename.</p> <p><b>Range:</b> 2 through 1000<br/> <b>Default:</b> 10 files</p> <p><b>flag</b>—Tracing operation to perform. To specify more than one tracing operation, include multiple <b>flag</b> statements. You can include the following flags:</p> <ul style="list-style-type: none"> <li>■ <b>all</b>—Log all operations</li> <li>■ <b>events</b>—Log important events</li> <li>■ <b>input</b>—Log script input data</li> <li>■ <b>offline</b>—Generate data for offline development</li> <li>■ <b>output</b>—Log script output data</li> <li>■ <b>rpc</b>—Log script RPCs</li> <li>■ <b>xslt</b>—Log the XSLT library</li> </ul> <p><b>no-world-readable</b>—Restrict file access to owner. This is the default.</p> |

**size** *size*—(Optional) Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named *trace-file* reaches this size, it is renamed and compressed to *trace-file.0.gz*. When *trace-file* again reaches its maximum size, *trace-file.0.gz* is renamed *trace-file.1.gz* and *trace-file* is renamed and compressed to *trace-file.0.gz*. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option and a filename.

**Syntax:** *xk* to specify KB, *xm* to specify MB, or *xg* to specify GB

**Range:** 10 KB through 1 GB

**Default:** 128 KB

**world-readable**—Enable unrestricted file access.

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

**Related Topics**

- Tracing Commit Script Processing on page 177
- Tracing Op Script Processing on page 305





## **Part 3**

# **Operation (Op) Scripts**

- Op Scripts Overview on page 289
- Writing Op Scripts on page 291
- Configuring and Executing Op Scripts on page 299
- Op Script Examples on page 311
- Summary of Op Script Configuration Statements on page 333



## Chapter 17

# Op Scripts Overview

This chapter includes the following topics:

- Op Script Programming Overview on page 289
- How Op Scripts Work on page 289

### Op Script Programming Overview

---

JUNOS operation (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 two application programming interfaces (APIs): the JUNOS Extensible Markup Language (XML) API and the JUNOScript API. Op scripts are executed by the JUNOS management (mgd) process.

Op scripts enable 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.

Op scripts are based on two application programming interfaces (APIs) to the JUNOS Software: the JUNOS XML API and the JUNOScript API, which are discussed in “JUNOScript API and JUNOS XML API Overview” 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 “XSLT Overview” on page 15. For more information about SLAX, see “SLAX Overview” on page 27.

### How Op Scripts Work

---

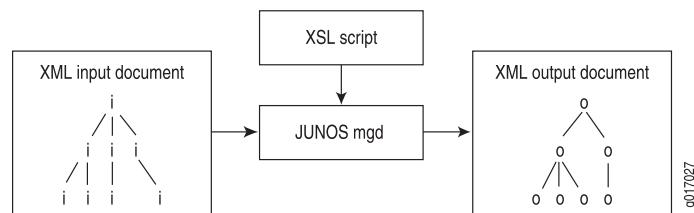
Op scripts execute router operational commands and inspect the resulting output. After inspection, op scripts can automatically correct errors within the router based on this output.

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 in Flash Memory” on page 302. 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 290 shows a high-level view of the flow of op script input and output.

**Figure 10: Op Script Input and Output**



## Chapter 18

# Writing Op Scripts

This chapter explains how to write operation (op) scripts and includes the following topics:

- Required Boilerplate for Op Scripts on page 291
- Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation on page 293
- Using RPCs and Operational Mode Commands in Op Scripts on page 294
- Declaring Arguments in Op Scripts on page 295
- Configuring Help Text for Op Scripts on page 298

### Required Boilerplate for Op Scripts

---

When you write operation (op) scripts, you use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided with the JUNOS Software. These tools include basic boilerplate that you must include in all commit scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make commit scripts easier to read and write, which you import from a file called `junos.xml`. For more information about the extension functions and templates, see “JUNOS Extension Functions Overview” on page 39.

Op 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.

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

#### XSLT Boilerplate for Op Scripts

```
1 <?xml version="1.0" standalone="yes"?>
2 <xsl:stylesheet version="1.0"
3 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4 xmlns:junos="http://xml.juniper.net/junos/*/junos"
5 xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6 xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
7 <xsl:import href="../import/junos.xml"/>
8
9 <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 referenced as `../import/junos.xml`, which ships as part of the JUNOS Software (in the file `/usr/libdata/cscript/import/junos.xml`). The `junos.xml` file contains a set of named templates you can call in your scripts. These named templates are discussed in “Importing the `junos.xml` File” on page 51.

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

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 Overview” 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>
```

Between Line 11 and Line 12, you can define additional XSLT templates that are called from within the `<xsl:template match="/">` template.

Line 12 closes the style sheet and the op script.

```
12 </xsl:stylesheet>
```

#### SLAX Boilerplate for Op Scripts

The corresponding SLAX boilerplate is as follows:

```
version 1.0;

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

match / {
 <op-script-results> {
 /*
 * Insert your code here
 */
 }
}
```

## Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation

In op scripts, you use tag elements from the JUNOS XML API to represent operational mode commands and output fields. For the JUNOS XML equivalent of commands and output fields, consult the *JUNOS XML API Operational Reference*.

You can also display the JUNOS XML tag elements for operational mode command output by directing the output from the command to the `| display xml` command:

```
user@host> command-string | display xml
```

For example:

```
user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
 <interface-information
 xmlns="http://xml.juniper.net/junos/10.0R10/junos-interface" junos:style="terse">
 <physical-interface>
 <name>dsc</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 </physical-interface>
 <physical-interface>
 <name>fxp0</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 <logical-interface>
 <name>fxp0.0</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 ...
```

## Using RPCs and Operational Mode Commands in Op Scripts

Most JUNOS operational mode commands have XML equivalents. These XML commands can be executed remotely using the *remote procedure call* (RPC) protocol. All operational mode commands that have XML equivalents are listed in the *JUNOS XML API Operational Reference*.

RPC and operational mode command use in op scripts is discussed in more detail in the following sections:

- Using RPCs in Op Scripts on page 294
- Using Operational Mode Commands in Op Scripts on page 294

### Using RPCs in Op Scripts

To use an RPC in an op script, include the RPC in a variable declaration, as shown in the following code snippet. This snippet is expanded and fully described in “Example: Customizing Output of the show interfaces terse Command in an Op Script” on page 316.

|                    |                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>XSLT Syntax</b> | <pre>&lt;xsl:variable name="rpc"&gt;   &lt;get-interface-information/&gt; # JUNOS RPC for the show interfaces command &lt;/xsl:variable&gt; &lt;xsl:variable name="out" select="jcs:invoke(\$rpc)"/&gt; ...</pre> |
| <b>SLAX Syntax</b> | <pre>var \$rpc = &lt;get-interface-information&gt;; var \$out = jcs:invoke(\$rpc);</pre>                                                                                                                          |

### Using Operational Mode Commands in Op Scripts

Some operational mode commands do not have XML equivalents. If a command is not listed in the *JUNOS XML API Operational Reference*, the command does not have an XML equivalent.

Another way to determine whether a command has an XML equivalent is to issue the command followed by the `| display xml` command:

```
user@host> operational-mode-command | display xml
```

If the output includes only tag elements like `<output>`, `<cli>`, and `<banner>`, the command might not have an XML equivalent. In the following example, the output indicates that the `show host` command has no XML equivalent:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
 <output>
 ...
 </output>
 <cli>
 <banner></banner>
 </cli>
```



```
</rpc-reply>
```



**NOTE:** For some commands that have an XML equivalent, the output of the piped | `display xml` command does not include tag elements other than `<output>`, `<cli>`, and `<banner>` only because the relevant feature is not configured. For example, the `show services cos statistics forwarding-class` command has an XML equivalent that returns output in the `<service-cos-forwarding-class-statistics>` response tag, but if the configuration does not include any statements at the `[edit class-of-service]` hierarchy level then there is no actual data for the `show services cos statistics forwarding-class | display xml` command to display. The output is something like this:

```
user@host> show services cos statistics forwarding-class | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/8.3I0/junos">
 <cli>
 <banner></banner>
 </cli>
</rpc-reply>
```

For this reason, the information in the *JUNOS XML API Operational Reference* is normally more reliable.

An op script can include commands that have no XML equivalent. Use the `<command>`, `<xsl:value-of>`, and `<output>` elements in the script, as shown in the following code snippet. This snippet is expanded and fully described in “Example: Displaying DNS Hostname Information in an Op Script” on page 313.

```
<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>
...
```

## Declaring Arguments in Op Scripts

There are two ways to declare arguments to an op script: by including XSLT or SLAX instructions in the script or by including statements in the JUNOS configuration. *Script-generated* and *configuration-generated* arguments have the same operational impact.

To declare arguments within a script, declare a global variable named `arguments`, containing `<argument>` tag elements. Within each `<argument>` tag element, include the required `<name>` tag element and the optional `<description>` tag element:

### XSLT Syntax

```
<xsl:variable name="arguments">
 <argument>
 <name>name</name>
```

```

 <description>name</description>
 </argument>
</xsl:variable>

```

**SLAX Syntax**

```

var $arguments = {
 <argument> {
 <name> "name";
 <description> "descriptive-text";
 }
}

```

To declare arguments in the configuration, include the **arguments** statement in the [edit system scripts op file *filename*] hierarchy level.

```

[edit system scripts op file filename]
arguments {
 argument-name {
 description descriptive-text;
 }
}

```

If you include the optional **<description>** tag element or the **description** statement, the text of the description appears in the command-line interface (CLI) as a help-text string to describe the purpose of the argument, as discussed in “Configuring Help Text for Op Scripts” on page 298.

In the operation script, you must include a corresponding parameter declaration for each argument. The parameter name must match the name of the argument:

```
<xsl:param name="name" />
```

The SLAX equivalent is:

```
param $name;
```

You can create a hidden argument by including the **<xsl:param name="*name*" />** instruction without listing the argument in the **arguments** variable or in the configuration.

After you declare an argument, you can use command completion to list available arguments:

```

user@host> op filename ?
Possible completions:
 argument-name description
 argument-name description

```

For each argument you include on the command-line, you must specify a corresponding value. To do this, include an ***argument-name*** and an ***argument-value*** when you execute the script with the **op *filename*** command:

```
user@host> op filename argument-name argument-value
```



**NOTE:** If you specify an argument that the script does not recognize, the script ignores the argument.

If you configure arguments by including the **arguments** statement in the configuration, any arguments that you declare directly in the script are still available, but are not listed among the **Possible completions** when you issue the **op filename ?** command:

If you declare all arguments in the script (and none in the configuration), then the arguments do appear in the **Possible completions** list. This is because the management (mgd) process populates the **Possible completions** list by first checking the configuration for arguments. The mgd process looks in the script for arguments only if no arguments are found in the configuration. Thus, if arguments are declared in the configuration, the arguments declared in the script become hidden in the CLI.

### Example: Declaring Arguments

Declare two arguments named **interface** and **protocol**. Execute the script, specifying the **ge-0/2/0.0** interface and the **inet** protocol as values for the arguments. For either method, you must declare corresponding script parameters:

#### Declaring Arguments in the Op Script (script1)

```
<xsl:param name="interface"/>
<xsl:param name="protocol"/>

<xsl:variable name="arguments">
 <argument>
 <name>interface</name>
 <description>Name of interface to display</description>
 </argument>
 <argument>
 <name>protocol</name>
 <description>Protocol to display (inet, inet6)</description>
 </argument>
</xsl:variable>
```

#### Declaring Arguments in the Configuration

```
[edit system scripts op]
file script1 {
 arguments {
 interface {
 description "Name of interface to display";
 }
 protocol {
 description "Protocol to display (inet, inet6)";
 }
 }
}
```

**Executing the Script**    user@host> **op script1 interface ge-0/2/0.0 protocol inet**

## Configuring Help Text for Op Scripts

---

You can provide help text to describe an op script and its arguments when the ? is used to list possible completions in the CLI. Include the **description** statement:

```
description descriptive-text;
```

You can include this statement at the following hierarchy levels:

- [edit system scripts op file *filename*]
- [edit system scripts op file *filename* arguments *argument-name*]

The following examples show the configuration and the resulting output.

### Examples: Configuring Help Text for Op Scripts

Configure help text for a script and display the resulting output:

```
[edit system scripts op]
user@host# set file interface.xml description "Test the interface"
user@host# commit
...
[edit system scripts op]
user@host# set file ?
Possible completions:
<name> Local filename of the script file
interface.xml Test the interface
```

Configure help text for a script's arguments and display the resulting output:

```
[edit system scripts op file interface.xml arguments]
user@host# set t1 description "Search for T1 interfaces"
user@host# set t3 description "Search for T3 interfaces"
user@host# commit
...
[edit system scripts op file interface.xml arguments]
user@host# set ?
Possible completions:
<name> Name of the argument
t1 Search for T1 interfaces
t3 Search for T3 interfaces
```

## Chapter 19

# Configuring and Executing Op Scripts

Operation (op) scripts allow you to automate network troubleshooting and network management. This chapter discusses command-line interface (CLI) configuration statements and operational mode commands for enabling and executing op scripts.

To configure op scripts, include the following statements at the [edit] hierarchy level:

```
system {
 scripts {
 op {
 file filename {
 arguments {
 argument-name {
 description descriptive-text;
 }
 }
 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;
 no-remote-trace;
 }
 }
 }
}
```

This chapter discusses the following topics:

- Implementing Op Scripts on page 300
- Enabling an Op Script and Defining a Script Alias on page 300
- Executing an Op Script on page 301
- Storing Op Scripts in Flash Memory on page 302
- Specifying a Master Source for an Op Script on page 302
- Updating an Op Script from the Master Source on page 303

- Updating an Op Script from an Alternate Location on page 304
- Converting an Op Script from XSLT to SLAX on page 304
- Converting an Op Script from SLAX to XSLT on page 305
- Tracing Op Script Processing on page 305

## Implementing Op Scripts

---

To use op scripts on a router or switch, follow these steps:

1. Write the op script.

For information on writing op scripts, see “Writing Op Scripts” on page 291. For examples, see “Op Script Examples” on page 311.

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; for information about setting the storage location for scripts, see “Storing Op Scripts in Flash Memory” on page 302. Only users who belong to the JUNOS **super-user** login class can access and edit files in these directories.



**NOTE:** If the router or switch has dual Routing Engines and you want to enable the op script to execute on both Routing Engines, you must copy the script to the `/var/db/scripts/op` or `/config/scripts/op` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

---

3. Enable the script by including the `file filename` statement at the `[edit system scripts op]` hierarchy level. For instructions, see “Enabling an Op Script and Defining a Script Alias” on page 300.
4. Issue the `commit` command.

After the commit operation completes, the op script can be executed on the router or switch. See “Executing an Op Script” on page 301.

## Enabling an Op Script and Defining a Script Alias

---

Operation (op) scripts are stored on a router or switch’s hard drive (in the `/var/db/scripts/op` directory) or flash drive (in the `/config/scripts/op` directory). Only users in the JUNOS superuser login class can access and edit files in these directories. For information about setting the storage location for scripts, see “Storing Op Scripts in Flash Memory” on page 302.



**NOTE:** If the router or switch has dual Routing Engines and you want to enable an op script to execute on both Routing Engines, you must copy the script to the `/var/db/scripts/op` or `/config/scripts/op` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

You must enable an op script before it can be executed. Include the `file filename` statement at the `[edit system scripts op]` hierarchy level, specifying the name of an XSLT or SLAX file containing an op script. Only users who belong to the JUNOS `super-user` login class can enable op scripts.

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

The filename of an op script written in SLAX must include the `.slax` extension for the script to be enabled and executed. No particular filename extension is required for op scripts written in XSLT, but we strongly recommend that you append the `.xsl` extension.

To determine which op scripts are currently active on the router, either list the contents of the `/var/run/scripts/op` directory or use the `show` command to display the files included at the `[edit system scripts op]` hierarchy level.

Optionally, you can define an alias for an op script and then specify either the filename or the alias when you execute the script. To define the alias, include the `command` statement at the `[edit system scripts op file filename]` hierarchy level:

```
[edit system scripts op]
file filename {
 command filename-alias;
}
```

## Executing an Op Script

Unlike commit scripts, operation (op) scripts do not execute during a commit operation. When you issue the `commit` command, op scripts enabled at the `[edit system scripts op]` hierarchy level are placed into system memory and enabled for execution. After the commit operation completes, you can execute an op script either by issuing the `op` operational mode command or automatically when a member of a JUNOS Software login class logs in to the CLI.

### Executing an Op Script by Issuing the `op` Command

To execute an op script from the CLI, issue the `op` operational mode command, specifying either the script filename or the alias defined by the `command` at the `[edit system scripts op file filename]` hierarchy level.

```
user@host> op filename-or-alias
```

## Executing an Op Script at Login

You can configure an op script to execute automatically when any user who belongs to a designated JUNOS login class logs in to the CLI. To associate an op script with a login class, include the `login-script script-filename` at the `[edit system login class class-name]` hierarchy level:

```
[edit system login]
class class-name {
 login-script script-filename;
}
```

The following example configures the `super-user-login.slax` op script to execute when any user who belongs to the `super-user` class logs in to the CLI (provided that the script has been enabled as discussed in “Enabling an Op Script and Defining a Script Alias” on page 300).

```
[edit system login]
class super-user {
 login-script super-user-login.slax;
}
```

## Storing Op Scripts in Flash Memory

By default, operation (op) scripts are stored in the `/var/db/scripts/op` directory on the router's hard drive. To store them in flash memory instead, include 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; op scripts are stored in the `/config/scripts/op` directory in flash memory. Changing the scripts' physical location has no effect on their operation.



**NOTE:** When you add or remove the `load-scripts-from-flash` statement in the configuration, you must manually move scripts from the hard drive to the flash drive, or vice versa, as appropriate. They are not moved automatically.

## 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 update the copy on each router where the op script is currently enabled.

To define the location of the master source file for 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;
```



- *filename*—Name of the op script.
- *url*—URL of the op script's master source file. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

The following example specifies an HTTP URL as the remote source for the `iso.xml` file:

```
[edit system scripts op]
file iso.xml {
 source http://my.example.com/pub/scripts/iso.xml;
}
```

Including the `source` statement in the configuration does not affect the local copy of the op script until you issue the `set refresh` command as described in “Updating an Op Script from the Master Source” on page 303. At that point, the master copy is retrieved from the specified URL and overwrites the local copy.

## Updating an Op Script from the Master Source

---

To update a single op script from its master source, issue the `set refresh` command at the `[edit system scripts op file filename]` hierarchy level. The master source must already be configured as described in “Specifying a Master Source for an Op Script” on page 302.

```
[edit system scripts op file filename]
user@host# set refresh
```

To update all enabled op scripts from their master sources, issue the `set refresh` command at the `[edit system scripts op]` hierarchy level:

```
[edit system scripts op]
user@host# set refresh
```

When you issue the `set refresh` command, the router immediately attempts to connect to the device that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local op scripts directory. If no master source for a script is defined, it is not updated and a warning is issued.

The update operation occurs as soon as you issue the `set refresh` command. The `refresh` statement is not added to the configuration. In other words, for this statement the `set` command behaves like an operational mode command, instead of adding a statement to the configuration.

If a platform has dual Routing Engines and you want the script to be updated 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.

## Updating an Op Script from an Alternate Location

---

In addition to updating an op script from the master source defined by the **source** statement at the [edit system scripts op file *filename*] hierarchy level, you also can update a script from an alternate location. This is convenient when, for example, the master source cannot be accessed due to network or other problems. To update a single op script from the alternate source, issue the **set refresh-from** command at the [edit system scripts op file *filename*] hierarchy level, specifying the location of the remote file:

```
[edit system scripts op file filename]
user@host# set refresh-from url
```

To update all enabled op scripts from the alternate source, issue the **set refresh-from** command at the [edit system scripts op] hierarchy level, specifying the location of the remote directory that houses the scripts:

```
[edit system scripts op]
user@host# set refresh-from url
```

At both hierarchy levels:

**url**—URL of the remote op script or directory. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

When you issue the **set refresh-from** command, the router attempts to connect to the device that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local op scripts directory. If no master source for a script is defined, it is not updated and a warning is issued.

The update operation occurs as soon as you issue the **set refresh-from** command. The **refresh-from** statement is not added to the configuration. In other words, for this statement, the **set** command behaves like an operational mode command, instead of adding a statement to the configuration.

If a platform has dual Routing Engines and you want the script to be updated on both Routing Engines, you must issue the **set refresh-from** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh-from** statement to update scripts on both Routing Engines.

## Converting an Op 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 “SLAX Overview” on page 27.

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

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/script1.slax
```

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. If you do not specify a filename for the destination file, it is named `SLAX-Conversion-Temp.xxxxx` where `xxxxx` is a randomly generated series of characters.

To convert a script from SLAX to XSLT, see “Converting an Op Script from SLAX to XSLT” on page 305.

## Converting an Op Script from SLAX to XSLT

---

To convert a SLAX script to XSLT, issue the `request system scripts convert slax-to-xslt` source *source-directory/source-filename* destination *destination-directory/<destination-filename>* operational mode command. 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/script1.xsl
```

When you issue this command, the `script1.slax` file remains unchanged in the `/var/db/scripts/op` directory and a new script called `script1.xsl` is added to the `/var/db/scripts/op` directory. If you do not specify a filename for the destination file, the file is named `SLAX-Conversion-Temp.xxxxx` where `xxxxx` is a randomly generated series of characters.

To convert a script from XSLT to SLAX, see “Converting an Op Script from XSLT to SLAX” on page 304.

## Tracing Op Script Processing

---

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

The default operation of op script tracing is to log important events in a file called `op-script.log` located in the `/var/log` directory. When the file `op-script.log` reaches 128 kilobytes (KB), it is renamed with a number 0 through 9 (in ascending order) appended to the end of the file and then compressed. The resulting files are `op-script.log.0.gz`, then `op-script.log.1.gz`, until there are 10 trace files. Then the oldest trace file (`op-script.log.9.gz`) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

This section discusses the following topics:

- Minimum Configuration for Enabling Traceoptions for Op Scripts on page 306
- Configuring Tracing of Op Scripts on page 307

### **Minimum Configuration for Enabling Traceoptions for Op Scripts**

If no op script trace options are configured, the simplest way to view the trace output of an op script is to configure the **output** trace flag and issue the **show log op-script.log | last** command. To do this, perform the following steps:

1. If you have not done so already, enable an op script by including the **file** statement at the **[edit system scripts op]** hierarchy level:

```
[edit system scripts op]
user@host# set file filename
```

2. Enable trace options by including the **traceoptions flag output** statement at the **[edit system scripts op]** hierarchy level:

```
[edit system scripts op]
user@host# set traceoptions flag output
```

3. Issue the **commit** command:

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

4. Display the resulting trace messages recorded in the file **/var/log/op-script.log** file. At the end of the log is the output generated by the op script you enabled in Step 1. To display the end of the log, issue the **show log op-script.log | last** operational mode command:

```
[edit]
user@host# run show log op-script.log | last
```

Table 15 on page 306 summarizes useful filtering commands that display selected portions of the **op-script.log** file.

**Table 15: Op Script Tracing Operational Mode Commands**

| Task                                                           | Command                                                 |
|----------------------------------------------------------------|---------------------------------------------------------|
| Display logging data associated with all op script processing. | <code>show log op-script.log</code>                     |
| Display processing for only the most recent operation.         | <code>show log op-script.log   last</code>              |
| Display processing for script errors.                          | <code>show log op-script.log   match error</code>       |
| Display processing for a particular script.                    | <code>show log op-script.log   match script-name</code> |

### Example: Minimum Configuration for Enabling Traceoptions for Op Scripts

Display the trace output of the op script file source-route.xml:

```
[edit]
system {
 scripts {
 op {
 file source-route.xml;
 traceoptions flag output;
 }
 }
}

[edit]
user@host# commit
[edit]
user@host# run show log op-script.log | last
```

### Configuring Tracing of Op Scripts

You cannot change the directory (`/var/log`) to which trace files are written. However, you can customize other trace file settings by including the following statements at the `[edit system scripts op traceoptions]` hierarchy level:

```
[edit system scripts op traceoptions]
file <filename> <files number> <size size> <world-readable | no-world-readable>;
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
no-remote-trace;
```

These statements are described in the following sections:

- Configuring the Op Script Log Filename on page 307
- Configuring the Number and Size of Op Script Log Files on page 308
- Configuring Access to Op Script Log Files on page 308
- Configuring the Op Script Trace Operations on page 308

### Configuring the Op Script Log Filename

By default, the name of the file that records trace output is `op-script.log`. You can specify a different name by including the `file` statement at the `[edit system scripts op traceoptions]` hierarchy level:

```
[edit system scripts op traceoptions]
file filename;
```

## Configuring the Number and Size of Op Script Log Files

By default, when the trace file reaches 128 KB in size, it is renamed and compressed to *filename.0.gz*, then *filename.1.gz*, and so on, until there are 10 trace files. Then the oldest trace file (*filename.9.gz*) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the [edit system scripts op traceoptions file <filename>] hierarchy level:

```
[edit system scripts op traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 640 KB and the maximum number of files to 20. When the file that receives the output of the tracing operation (*filename*) reaches 640 KB, it is renamed and compressed to *filename.0.gz*, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename.0.gz* is renamed *filename.1.gz* and *filename* is renamed and compressed to *filename.0.gz*. This process repeats until there are 20 trace files. Then the oldest file (*filename.19.gz*) is overwritten.

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

If you set either a maximum file size or a maximum number of trace files, you also must specify the other parameter and a filename.

## Configuring Access to Op Script Log Files

By default, access to the op script log file is restricted to the owner. You can manually configure access by including the *world-readable* or *no-world-readable* statement at the [edit system scripts op traceoptions file <filename>] hierarchy level.

```
[edit system scripts op traceoptions file <filename>]
(world-readable | no-world-readable);
```

The *no-world-readable* statement restricts op script log access to the owner. The *world-readable* statement enables unrestricted access to the op script log file.

## Configuring the Op Script Trace Operations

By default, only important events are logged. You can configure the trace operations to be logged by including the following statements at the [edit system scripts op traceoptions] hierarchy level:

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

Table 16 on page 309 describes the meaning of the op script tracing flags.

**Table 16: 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 20

# Op Script Examples

This chapter provides sample op scripts that run commands and customize output. This chapter includes the following topics:

- Example: Restarting an FPC in an Op Script on page 311
- Example: Displaying DNS Hostname Information in an Op Script on page 313
- Example: Customizing Output of the show interfaces terse Command in an Op Script on page 316
- Example: Finding LSPs to Multiple Destinations in an Op Script on page 326
- Example: Importing and Exporting Files in an Op Script on page 330

### Example: Restarting an FPC in an Op Script

---

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 than using a remote procedure call (RPC). For more information, see “Using RPCs and Operational Mode Commands in Op Scripts” on page 294.

#### 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.xml";

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 the ex-fpc Script**

To test the **ex-fpc** script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Restarting an FPC in an Op Script” on page 311 into a text file, name the file **ex-fpc.xml** or **ex-fpc.slax** as appropriate, and copy it to the **/var/db/scripts/op** directory on the router.

2. In configuration mode, include the `file ex-fpc.extension` statement at the `[edit system scripts op]` hierarchy level, substituting `.slax` or `.xsl` for *extension* as appropriate.

```
[edit system scripts op]
file ex-fpc.(slax | xsl);
```

3. Issue the `commit and-quit` command.

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

When you issue the `op ex-fpc slot number` operational command, output like the following appears:

```
user@host> op ex-fpc slot 0
Restarting the FPC in slot 0. To verify, issue the "show chassis fpc" command.
```

## Example: Displaying DNS Hostname Information in an Op Script

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 than using a remote procedure call (RPC). For more information, see “Using RPCs and Operational Mode Commands in Op Scripts” on page 294.

The script is provided in two distinct versions, one using the `<xsl:choose>` element and the other the `jcs:first-of()` function. Both versions accept the same argument and produce the same output.

### XSLT Syntax Using the <xsl:choose> Element

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

 <xsl:variable name="arguments">
 <argument>
 <name>dns</name>
 <description>Name or IP address of a host</description>
 </argument>
 </xsl:variable>
 <xsl:param name="dns"/>
 <xsl:template match="/">
 <op-script-results>
 <xsl:variable name="query">
 <xsl:choose>
 <xsl:when test="$dns">
 <command>
```

```

 <xsl:value-of select="concat('show host ', $dns)"/>
 </command>
 </xsl:when>
 <xsl:when test="$hostname">
 <command>
 <xsl:value-of select="concat('show host ', $hostname)"/>
 </command>
 </xsl:when>
 </xsl:choose>
</xsl:variable>
<xsl:variable name="result" select="jcs:invoke($query)"/>
<xsl:variable name="host" select="$result"/>
<output>
 <xsl:value-of select="concat('Name: ', $host)"/>
</output>
</op-script-results>
</xsl:template>
</xsl:stylesheet>

```

#### **XSLT Syntax Using the jcs:first-of() Function**

```

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

 <xsl:variable name="arguments">
 <argument>
 <name>dns</name>
 <description>Name or IP address of a host</description>
 </argument>
 </xsl:variable>
 <xsl:param name="dns"/>
 <xsl:template match="/">
 <op-script-results>
 <xsl:variable name="target" select="jcs:first-of($dns, $hostname)"/>
 <xsl:variable name="query">
 <command>
 <xsl:value-of select="concat('show host ', $target)"/>
 </command>
 </xsl:variable>
 <xsl:variable name="result" select="jcs:invoke($query)"/>
 <xsl:variable name="host" select="$result"/>
 <output>
 <xsl:value-of select="concat('Name: ', $host)"/>
 </output>
 </op-script-results>
 </xsl:template>
</xsl:stylesheet>

```

#### **SLAX Syntax Using the <xsl:choose> Element**

```

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

```

```

import "../import/junos.xml";

var $arguments = {
 <argument> {
 <name> "dns";
 <description> "Name or IP address of a host";
 }
}
param $dns;
match / {
 <op-script-results> {
 var $query = {
 if ($dns) {
 <command> 'show host ' _ $dns;
 } else if ($hostname) {
 <command> 'show host ' _ $hostname;
 }
 }
 var $result = jcs:invoke($query);
 var $host = $result;
 <output> 'Name: ' _ $host;
 }
}

```

#### SLAX Syntax Using the jcs:first-of() Function

```

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

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 the ex-hostname Script

To test the ex-hostname script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Displaying DNS Hostname Information in an Op Script” on page 313 into a text file, name the file `ex-hostname.xml` or `ex-hostname.slax` as appropriate, and copy it to the `/var/db/scripts/op` directory on the router.
2. In configuration mode, include the file `ex-hostname.extension` statement at the `[edit system scripts op]` hierarchy level, substituting `.slax` or `.xml` for `extension` as appropriate.

```
[edit system scripts op]
file ex-hostname.(slax | xml);
```

3. Issue the `commit and-quit` command.

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

When you issue the `op ex-hostname` operational mode command without the `dns` option, DNS information is displayed for the local router:

```
user@host> op ex-hostname
Name:
this-router has address 10.168.71.246
```

When you issue the `op ex-hostname dns hostname` command, DNS information is displayed for the specified router:

```
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, DNS information is displayed for the specified address:

```
user@host> op ex-hostname dns 10.168.71.249
Name:
249.71.168.10.IN-ADDR.ARPA domain name pointer router1
```

## Example: Customizing Output of the show interfaces terse Command in an Op Script

By default, the layout of the `show interfaces terse` command looks like this:

```
user@host> show interfaces terse
```

| Interface | Admin | Link | Proto | Local                  | Remote |
|-----------|-------|------|-------|------------------------|--------|
| dsc       | up    | up   |       |                        |        |
| fxp0      | up    | up   |       |                        |        |
| fxp0.0    | up    | up   | inet  | 192.168.71.246/21      |        |
| fxp1      | up    | up   |       |                        |        |
| fxp1.0    | up    | up   | inet  | 10.0.0.4/8             |        |
|           |       |      | inet6 | fe80::200:ff:fe00:4/64 |        |

```

 fec0::10:0:0:4/64
 tnp 4
gre up up
ipip up up
lo0 up up
lo0.0 up up inet 127.0.0.1 --> 0/0
lo0.16385 up up inet inet6 fe80::2a0:a5ff:fe12:2f04

lsi up up
mtun up up
pimd up up
pime up up
tap up up

```

In JUNOS XML, the output fields are represented as follows:

```

user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0RIO/junos">
 <interface-information
 xmlns="http://xml.juniper.net/junos/10.0RIO/junos-interface" junos:style="terse">
 <physical-interface>
 <name>dsc</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 </physical-interface>
 <physical-interface>
 <name>fxp0</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 <logical-interface>
 <name>fxp0.0</name>
 <admin-status>up</admin-status>
 <oper-status>up</oper-status>
 ... Remainder of output omitted for brevity ...
 </interface-information>
</rpc-reply>

```

**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.xml"/>

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'
36 or $protocol='mpls' or $protocol='tnp'">
37 <xsl:for-each select="$out/physical-interface/
38 logical-interface[address-family/address-family-name = $protocol]">
39 <xsl:call-template name="intf"/>
40 </xsl:for-each>
41 </xsl:when>
42 <xsl:when test="$protocol">
43 <xnm:error>
44 <message>
45 <xsl:text>invalid protocol: </xsl:text>
46 <xsl:value-of select="$protocol"/>
47 </message>
48 </xnm:error>
49 </xsl:when>
50 <xsl:otherwise>
51 <xsl:for-each
52 select="$out/physical-interface/logical-interface">
53 <xsl:call-template name="intf"/>
54 </xsl:for-each>
55 </xsl:otherwise>
56 </xsl:choose>
57 </interface-information>
58 </op-script-results>
59 </xsl:template>
60 <xsl:template name="intf">
61 <xsl:variable name="status">
62 <xsl:choose>
63 <xsl:when test="admin-status='up' and oper-status='up'">
64 <xsl:text> </xsl:text>
65 </xsl:when>
66 <xsl:when test="admin-status='down'">
67 <xsl:text>offline</xsl:text>
68 </xsl:when>
69 <xsl:when test="oper-status='down' and ../admin-status='down'">
70 <xsl:text>p-offline</xsl:text>
71 </xsl:when>
72 <xsl:when test="oper-status='down' and ../oper-status='down'">
73 <xsl:text>p-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
99 select="address-family[address-family-name = $protocol]"/>
100 </xsl:when>
101 <xsl:otherwise>
102 <xsl:copy-of select="address-family"/>
103 </xsl:otherwise>
104 </xsl:choose>
105 </logical-interface>
106 </xsl:template>
107 </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 “Required Boilerplate for Op Scripts” on page 291.

```

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.xml"/>
...
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'
36 or $protocol='mpls' or $protocol='tnp'">

```

```

37 logical-interface[address-family/address-family-name = $protocol]">
38 <xsl:call-template name="intf"/>
39 </xsl:for-each>
40 </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
50 select="$out/physical-interface/logical-interface">
51 <xsl:call-template name="intf"/>
52 </xsl:for-each>
53 </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
99 select="address-family[address-family-name = $protocol]"/>
100 </xsl:when>
101 <xsl:otherwise>
102 <xsl:copy-of select="address-family"/>
103 </xsl:otherwise>
104 </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 the ex-interface Script

To test the `ex-interface` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Customizing Output of the show interfaces terse Command in an Op Script” on page 316 into a text file, name the file `ex-interface.xml` or `ex-interface.slax` as appropriate, and copy it to the `/var/db/scripts/op` directory on the router.
2. In configuration mode, include the file `ex-interface.extension` statement at the `[edit system scripts op]` hierarchy level, substituting `.slax` or `.xml` for `extension` as appropriate.

```

[edit system scripts op]
file ex-interface.(slax | xml);

```

3. Issue the commit and-quit command.

```

[edit]
user@host# commit and-quit

```

Issue the `show interfaces terse` and `op ex-interface` operational commands and compare the output.

```

user@host> show interfaces terse

```

| Interface | Admin | Link | Proto | Local                    | Remote  |
|-----------|-------|------|-------|--------------------------|---------|
| dsc       | up    | up   |       |                          |         |
| fxp0      | up    | up   |       |                          |         |
| fxp0.0    | up    | up   | inet  | 192.168.71.246/21        |         |
| fxp1      | up    | up   |       |                          |         |
| fxp1.0    | up    | up   | inet  | 10.0.0.4/8               |         |
|           |       |      | inet6 | fe80::200:ff:fe00:4/64   |         |
|           |       |      |       | fec0::10:0:0:4/64        |         |
|           |       |      | tnp   | 4                        |         |
| gre       | up    | up   |       |                          |         |
| ipip      | up    | up   |       |                          |         |
| lo0       | up    | up   |       |                          |         |
| lo0.0     | up    | up   | inet  | 127.0.0.1                | --> 0/0 |
| lo0.16385 | up    | up   | inet  |                          |         |
|           |       |      | inet6 | fe80::2a0:a5ff:fe12:2f04 |         |

```

lsi up up
mtun up up
pimd up up
pime up up
tap up up

user@host> op 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

```

## Example: Finding LSPs to Multiple Destinations in an Op Script

This sample script 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, ') -> ', $lsp/name)"/>
 </output>
 </xsl:when>
 <xsl:otherwise>
 <xsl:variable name="name"
 select="jcs:hostname($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>

```

```

 </output>
 </xsl:otherwise>
 </xsl:choose>
 </xsl:for-each>
</xsl:otherwise>
</xsl:choose>
</xsl:otherwise>
</xsl:choose>
</op-script-output>
</xsl:template>
</xsl:stylesheet>

```

**SLAX Syntax**

```

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

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 the ex-lsp Script

To test the `ex-lsp` script, perform the following steps:

1. Copy the XSLT or SLAX script from “Example: Finding LSPs to Multiple Destinations in an Op Script” on page 326 into a text file, name the file **ex-lsp.xml** or **ex-lsp.slax** as appropriate, and copy it to the **/var/db/scripts/op** directory on the router.
2. In configuration mode, include the **file ex-lsp.extension** statement at the **[edit system scripts op]** hierarchy level, substituting **.slax** or **.xml** for **extension** as appropriate.

```
[edit system scripts op]
file ex-lsp.(slax | xsl);
```

3. Issue the `commit and-quit` command.

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

When you issue the `op ex-isp address address` operational mode command, output like the following appears:

```
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
```

## Example: Importing and Exporting Files in an Op Script

---

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 21

# Summary of Op Script Configuration Statements

This chapter describes each configuration statement for operation (op) scripts. The statements are organized alphabetically.

### arguments

---

**Syntax**    arguments {  
              *argument-name* {  
                  description *descriptive-text*;  
              }  
          }

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

**Release Information**    Statement introduced in JUNOS Release 7.6.

**Description**    For JUNOS op scripts, configure command-line arguments to the script.

**Options**    *argument-name*—The name of a command-line argument to an op script.  
  
The remaining statement is explained separately.

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

**Related Topics**    Declaring Arguments in Op Scripts on page 295

## command

---

|                                 |                                                                                                                                                                          |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | command <i>filename-alias</i> ;                                                                                                                                          |
| <b>Hierarchy Level</b>          | [edit system scripts op file <i>filename</i> ]                                                                                                                           |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.6.                                                                                                                               |
| <b>Description</b>              | 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. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                              |
| <b>Related Topics</b>           | Enabling an Op Script and Defining a Script Alias on page 300                                                                                                            |

## description

---

|                                 |                                                                                                                                                                                            |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | description <i>descriptive-text</i> ;                                                                                                                                                      |
| <b>Hierarchy Level</b>          | [edit system scripts op file <i>filename</i> ]<br>[edit system scripts op file <i>filename</i> arguments <i>argument-name</i> ]                                                            |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.6.                                                                                                                                                 |
| <b>Description</b>              | For JUNOS op scripts, provide a help-text string that appears in the command-line interface (CLI).                                                                                         |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ file (Op Scripts)</li> <li>■ Declaring Arguments in Op Scripts on page 295</li> <li>■ Configuring Help Text for Op Scripts on page 298</li> </ul> |



# file (Op Scripts)

---

|                          |                                                                                                                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax                   | <pre>file filename {   arguments {     argument-name {       description descriptive-text;     }   }   command filename-alias;   description descriptive-text;   refresh;   refresh-from url;   source url; }</pre>        |
| 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                  | <p><i>filename</i>—The name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an op script.</p> <p>The statements are explained separately.</p> |
| Required Privilege Level | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                     |
| Related Topics           | ■ Enabling an Op Script and Defining a Script Alias on page 300                                                                                                                                                            |

**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> <world-readable | no-world-readable>;
 flag flag;
 no-remote-trace;
 }
 }

```

**Hierarchy Level** [edit system scripts]

**Release Information** Statement introduced in JUNOS Release 7.6.

**Description** For JUNOS op scripts, configure an operation scripting mechanism.

**Options** The statements are explained separately.

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

**Related Topics** ■ Implementing Op Scripts on page 300

## refresh (Op Scripts)

---

|                                 |                                                                                                                                                                                                                                                                                   |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh;                                                                                                                                                                                                                                                                          |
| <b>Hierarchy Level</b>          | [edit system scripts op],<br>[edit system scripts op file <i>filename</i> ]                                                                                                                                                                                                       |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.6.                                                                                                                                                                                                                                        |
| <b>Description</b>              | For JUNOS op scripts, overwrite the local copy of all enabled op scripts or a single enabled script located in the <code>/var/db/scripts/op</code> directory with the copy located at the source URL, specified in the <code>source</code> statement at the same hierarchy level. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                       |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh-from (Op Scripts)</li> <li>■ source (Op Scripts)</li> <li>■ Updating an Op Script from the Master Source on page 303</li> </ul>                                                                                                  |

## refresh-from (Op Scripts)

---

|                                 |                                                                                                                                                                                                                                                                |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh-from <i>url</i> ;                                                                                                                                                                                                                                      |
| <b>Hierarchy Level</b>          | [edit system scripts op],<br>[edit system scripts op file <i>filename</i> ]                                                                                                                                                                                    |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.6.                                                                                                                                                                                                                     |
| <b>Description</b>              | For JUNOS op scripts, overwrite the local copy of all enabled op scripts or a single enabled script located in the <code>/var/db/scripts/op</code> directory with the copy located at a URL other than the URL specified in the <code>source</code> statement. |
| <b>Options</b>                  | <i>url</i> —Source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.                                                                                                                       |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                    |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh (Op Scripts)</li> <li>■ source (Op Scripts)</li> <li>■ Updating an Op Script from an Alternate Location on page 304</li> </ul>                                                                                |

## scripts

---

**See** scripts

## source (Op Scripts)

---

**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, the local copy is overwritten by the version stored at the specified URL.

**Options** *url*—Master source file for an op script specified as an HTTP URL, FTP URL, or scp-style remote file specification.

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

**Related Topics**

- refresh (Op Scripts)
- refresh-from (Op Scripts)
- Specifying a Master Source for an Op Script on page 302
- Updating an Op Script from the Master Source on page 303

## traceoptions

---

**See** traceoptions (Commit and Op Scripts)

## **Part 4**

# **Event Policy**

- Event Policy Overview on page 341
- Configuring Event Policy on page 345
- Event Policy Examples on page 371
- Summary of Event Policy Configuration Statements on page 383



## Chapter 22

# Event Policy Overview

This chapter discusses the following topics:

- Event Notifications and Policies Overview on page 341
- How Event Policies Work on page 342

## Event Notifications and Policies Overview

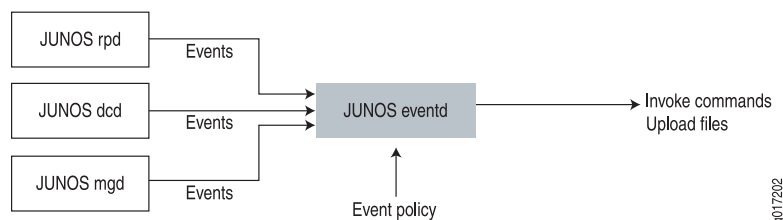
To diagnose a fault or error condition on a routing platform, you need relevant information about the state of the platform. You can derive state information from *event notifications*. Event notifications are system log messages and SNMP traps. A JUNOS Software process called the *event process* (eventd) 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 a log file, invoke a JUNOS command, or invoke an event script. When an event script is invoked, event details are passed to the event script in the form of XML inputs.

Figure 11 on page 341 shows how the event process (eventd) interacts with other JUNOS Software processes.

**Figure 11: Interaction of eventd Process 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 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*.

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 an SNMP trap.



## Chapter 23

# Configuring Event Policy

Event policies can monitor specific events, create log files, invoke JUNOS Software commands, and invoke event scripts. This chapter discusses the command-line interface (CLI) statements for configuring event policies.

To configure event policy, include the following statements at the [edit event-options] hierarchy level:

```
[edit event-options]
destinations {
 destination-name {
 archive-sites {
 url <password password>;
 }
 transfer-delay seconds;
 }
}
generate-event event-name {
 time-interval seconds;
 time-of-day hh:mm:ss;
}
policy policy-name {
 attributes-match {
 event1.attribute-name equals event2.attribute-name;
 event.attribute-name matches regular-expression;
 event1.attribute-name starts-with event2.attribute-name;
 }
 events [events];
 within seconds {
 events [events];
 not events [events];
 trigger (on | after | until) event-count;
 }
 then {
 event-script script-name {
 arguments {
 argument-name argument-value;
 }
 }
 destination destination-name {
 retry-count number retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 }
}
```

```

 user-name name;
 }
 execute-commands {
 commands {
 "command";
 }
 destination destination-name {
 retry-count number retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 user-name username;
 }
 ignore;
 raise-trap;
 upload filename (filename | committed) destination destination-name {
 retry-count number retry-interval seconds;
 transfer-delay seconds;
 user-name username;
 }
}
}
traceoptions {
 file <filename> <files number> <match regular-expression> <size size>
 <world-readable | no-world-readable>;
 flag flag;
 no-remote-trace;
}

```

This chapter discusses the following topics:

- Using Correlated Events to Trigger an Event Policy on page 347
- Representing the Correlating Event in an Event Policy on page 349
- Triggering an Event Policy Based on Event Count on page 350
- Using Regular Expressions to Refine the Set of Events That Trigger a Policy on page 350
- Generating Internal Events to Trigger Event Policies on page 351
- Using Nonstandard System Log Messages to Trigger Event Policies on page 352
- Defining Destinations for File Archiving by Event Policies on page 353
- Configuring an Event Policy to Upload Files on page 354
- Configuring the Delay Before Files Are Uploaded by an Event Policy on page 355
- Configuring an Event Policy to Retry the File Upload Action on page 356
- Configuring an Event Policy to Execute Operational Mode Commands on page 357
- Executing Event Scripts in an Event Policy on page 360
- Configuring Event Policies to Ignore an Event on page 365
- Changing the User Privilege Level for an Event Policy Action on page 366

- Configuring Event Policies to Raise SNMP Traps on page 366
- Tracing Event Policy Processing on page 367

## Using Correlated Events to Trigger an Event Policy

You can configure a policy that correlates two or more events. If the correlated events occur as specified, they cause particular actions to be taken. For example, you might want to issue certain operational mode commands when a `UI_CONFIGURATION_ERROR` event is generated within five minutes (300 seconds) after a `UI_COMMIT_PROGRESS` event. As another example, you might want to upload a particular file if a `DCD_INTERFACE_DOWN` event is generated two times within a 60-second interval.

To configure a policy that correlates events, include the following statements at the `[edit event-options]` hierarchy level:

```
[edit event-options]
policy policy-name {
 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];
 within seconds {
 events [events];
 not events [events];
 trigger (on | after | until) event-count;
 }
}
```

In the `events` statement, you can list multiple events. To view a list of the events that can be referenced in an event policy, issue the `set event-options policy policy-name events ?` configuration mode command:

```
user@host# set event-options policy policy-name events ?
Possible completions:
<event>
[Open a set of values
acct_accounting_ferror
acct_accounting_fopen_error
...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the `set event-options policy policy-name events ?` command. For information about referencing these system log messages in your event policies, see “Using Nonstandard System Log Messages to Trigger Event Policies” on page 352.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events to Trigger Event Policies” on page 351.

The actions configured in the `then` statement are executed only if certain conditions are met, which you specify in the `within` and `attributes-match` statements.

You can configure a policy that is executed only if a specified event occurs within a specified time interval after another event. You do this by including the **within seconds events** statement. The policy is executed only if one or more of the events in the first **events** statement occur within a configured number of seconds after one or more of the events in the **within seconds events** statement. The number of seconds can be from 60 through 604,800. The **not** statement causes the policy to be executed only if the events do not occur within the configured time interval.

For example, the following policy is executed if *event3*, *event4*, or *event5* occurs within 60 seconds after *event1* or *event2* occurs:

```
[edit event-options]
policy 1 {
 events [event3 event4 event5];
 within 60 events [event1 event2];
 then {
 ...
 }
}
```

The **attributes-match** statement correlates two events as follows:

- *event1.attribute-name* equals *event2.attribute-name*—Execute the policy only if the specified attribute of *event1* equals the specified attribute of *event2*.
- *event.attribute-name* matches *regular-expression*—Execute the policy only if the specified attribute of *event* matches a regular expression. For more information, see “Using Regular Expressions to Refine the Set of Events That Trigger a Policy” on page 350.
- *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 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
```

|                      |                                                            |
|----------------------|------------------------------------------------------------|
| <code>find</code>    | Search for first occurrence of pattern                     |
| <code>hold</code>    | Hold text without exiting the <code>--More--</code> prompt |
| <code>last</code>    | Display end of output only                                 |
| <code>match</code>   | Show only text that matches a pattern                      |
| <code>no-more</code> | Don't paginate output                                      |
| <code>request</code> | Make system-level requests                                 |
| <code>resolve</code> | Resolve IP addresses                                       |
| <code>save</code>    | Save output text to file                                   |
| <code>trim</code>    | 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 “Example: Ignoring Events Based on Receipt of Other Events” on page 378, “Example: Correlating Events Based on Event Attributes” on page 379, and “Controlling Event Policy Using a Regular Expression” on page 380.

## Representing the Correlating Event in an Event Policy

---

As described in “Configuring an Event Policy to Execute Operational Mode Commands” on page 357, the double dollar sign (`$$`) notation represents the event that is triggering a policy. Triggering events are those that you configure at the `[edit event-options policy policy-name events]` hierarchy level.

As described in “Using Correlated Events to Trigger an Event Policy” on page 347, you can configure a policy that is executed only if a specified event occurs within a specified time interval after another event. You do this by including the `within seconds events` statement at the `[edit event-options policy policy-name]` hierarchy level:

```
[edit event-options policy policy-name]
events [events];
within seconds events [events];
```

The policy is executed only if one or more of the events at the `[edit event-options policy policy-name events]` hierarchy level occur within a configured number of seconds after one or more of the events in the `within seconds events` statement.

For correlating events, the single dollar sign with the event name (`$event`) notation represents the most recent event that matches the event name. The dollar sign with

the asterisk (\$\*) notation represents the most recent event that matches any of the correlating events.

For a configuration example, see “Example: Representing the Correlating Event in an Event Policy” on page 374.

## Triggering an Event Policy Based on Event Count

---

You can configure an event policy to be triggered if an event or set of events occurs a specified number of times within a specified time period.

To do this, include the optional **trigger** statement at the [edit event-options policy *policy-name* within seconds] hierarchy level:

```
[edit event-options policy policy-name within seconds]
trigger (after | on | until) event-count;
```

The software counts the number of times the triggering event occurs. A triggering event can be any event configured at the [edit event-options policy *policy-name* events] hierarchy level. You can configure the following options:

- **after *event-count***—The policy is executed when the number of matching events received equals *event-count* plus one.
- **on *event-count***—The policy is executed when the number of matching events received equals *event-count*.
- **until *event-count***—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals *event-count*.

For configuration examples, see “Examples: Triggering a Policy Based on Event Count” on page 377.

## Using Regular Expressions to Refine the Set of Events That Trigger a Policy

---

You can use regular expression matching to specify more exactly which events cause a policy to be executed.

To specify the text string that must appear in an event attribute for the policy to be executed, include the **matches** statement at the [edit event-options policy *policy-name* attributes-match] hierarchy level, and specify the regular expression which the event attribute must match:

```
[edit event-options policy policy-name attributes-match]
event.attribute-name matches regular-expression;
```

When you specify the regular expression, use the notation defined in POSIX Standard 1003.2 for extended (modern) UNIX regular expressions. Explaining regular expression syntax is beyond the scope of this document. Table 17 on page 351 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 17: Regular Expression Operators for the matches Statement**

| Operator                     | Matches                                                                                                                                                                                                                              |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| . (period)                   | One instance of any character except the space.                                                                                                                                                                                      |
| * (asterisk)                 | Zero or more instances of the immediately preceding term.                                                                                                                                                                            |
| + (plus sign)                | One or more instances of the immediately preceding term.                                                                                                                                                                             |
| ? (question mark)            | Zero or one instance of the immediately preceding term.                                                                                                                                                                              |
| (pipe)                       | One of the terms that appear on either side of the pipe operator.                                                                                                                                                                    |
| ! (exclamation point)        | Any string except the one specified by the expression, when the exclamation point appears at the start of the expression. Use of the exclamation point is specific to the JUNOS Software.                                            |
| ^ (caret)                    | The start of a line, when the caret appears outside square brackets.<br><br>One instance of any character that does not follow it within square brackets, when the caret is the first character inside square brackets.              |
| \$ (dollar sign)             | The end of a line.                                                                                                                                                                                                                   |
| [ ] (paired square brackets) | One instance of one of the enclosed alphanumeric characters. To indicate a range of characters, use a hyphen ( - ) to separate the beginning and ending characters of the range. For example, [a-z0-9] matches any letter or number. |
| ( ) (paired parentheses)     | One instance of the evaluated value of the enclosed term. Parentheses are used to indicate the order of evaluation in the regular expression.                                                                                        |

For a configuration example, see “Controlling Event Policy Using a Regular Expression” on page 380.

## Generating Internal Events to Trigger Event Policies

*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 “Example: Generating an Internal Event Every Hour” on page 380 and “Example: Generating an Internal Event at Midnight” on page 381.

## Using Nonstandard System Log Messages to Trigger Event Policies

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. These system log messages have an event ID and a **message** attribute. Event IDs are based on the origin of the message, as shown in Table 18 on page 352.

**Table 18: 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 router, messages from a line-card chassis (LCC)   |
| SCC       | On a TX Matrix router, messages from a switch-card chassis (SCC) |

To base your event policy on the event types shown in Table 18 on page 352, include the `events event-id` statement and the `attributes-match` statement with the `event-id.message` matches "message" attribute at the `[edit event-options policy policy-name]` hierarchy level:

```
[edit event-options policy policy-name]
events event-id;
attributes-match {
 event-id.message matches "message";
}
```

For a configuration example, see “Example: Using Nonstandard System Log Messages to Trigger an Event Policy” on page 381.

## Defining Destinations for File Archiving by Event Policies

When the action in an event policy generates output files, you might want to save them for later analysis. Similarly, you might want to save system files (such as system log or core files) from the time an event occurs. You must configure one or more *destinations* to which files can be uploaded for archiving. To define destinations, include the `destinations` statement at the `[edit event-options]` hierarchy level:

```
[edit event-options]
destinations {
 destination-name {
 archive-sites {
 url <password password>;
 }
 transfer-delay seconds;
 }
}
```

You can then reference configured destinations in an event policy. For information about referencing destinations, see “Configuring an Event Policy to Upload Files” on page 354 and “Configuring an Event Policy to Execute Operational Mode Commands” on page 357.

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. For more information, see “Configuring the Delay Before Files are Uploaded by an Event Policy” on page 355.

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 “Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval” on page 371.

## Configuring an Event Policy to Upload 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 (filename | committed) destination destination-name {
 retry-count number retry-interval seconds;
 transfer-delay seconds;
 user-name username;
 }
 }
}
```

When an event policy uploads files for analysis, the files are named and time-stamped in the following format to ensure unique filenames:

*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 `rpdc-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:

```
[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
<event>
[Open a set of values
acct_accounting_ferror
acct_accounting_fopen_error
...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the `set event-options policy policy-name events ?` command. For information about referencing these system log messages in your event policies, see “Using Nonstandard System Log Messages to Trigger Event Policies” on page 352.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events to Trigger Event Policies” on page 351.

If desired, you can include multiple `upload` statements, one for each type of file to be archived. In the `filename` statement, specify a file or multiple files to be uploaded. You can specify multiple files with one `filename` configuration statement (sometimes called *filename globbing*). For example, to upload all files that are located in the `/var/log` directory and that start with the `messages` string, include the following statement:

```
upload filename /var/log/messages*;
```

To upload the committed configuration file, include the `upload filename committed destination destination-name` statement at the `[edit event-options policy policy-name then]` hierarchy level:

```
[edit event-options policy policy-name then]
upload filename committed destination destination-name;
```

For the `destination` statement, specify a destination name that you configured at the `[edit event-options destinations]` hierarchy level. For more information, see “Defining Destinations for File Archiving by Event Policies” on page 353.

## Configuring the Delay Before Files Are Uploaded by an Event Policy

A transfer delay allows you to specify the number of seconds the event process (eventd) waits before beginning to upload a file or multiple files. A transfer delay allows you to ensure that a large file, such as a core file, is completely generated before the upload begins.

As described in “Defining Destinations for File Archiving by Event Policies” 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* | committed) destination *destination*]

If you configure a transfer delay at the [edit event-options destinations *destination-name*] hierarchy level, and you also configure a transfer delay for the event policy action, the resulting transfer delay is the sum of the two:

```
Total transfer-delay =
transfer-delay (destination) + transfer-delay (event-policy-action)
```

For configuration examples, see “Examples: Assigning a Transfer Delay to an Event Policy Action” on page 372.

## Configuring an Event Policy to Retry the File Upload Action

Transient network problems can cause a file upload operation to fail. When this happens, you might want to retry the file upload operation. By default, if the file upload operation fails for any reason, the event policy does not retry the upload operation.

To configure the policy to retry a file upload operation, include the optional `retry-count` and `retry-interval` statements:

```
retry-count number retry-interval seconds;
```

You can include these statements at the following hierarchy levels:

- [edit event-options policy *policy-name* then event-script *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* | committed) 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 “Examples: Retrying the File Upload Action” on page 375.

## Configuring an Event Policy to Execute 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:

```

[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
<event>
[Open a set of values
acct_accounting_ferror
acct_accounting_fopen_error
...

```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see “Using Nonstandard System Log Messages to Trigger Event Policies” on page 352.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events to Trigger Event Policies” on page 351.

In the **commands** statement, you can issue multiple operational mode commands upon receipt of a specific event. Enclose each command in quotation marks (“ ”). The eventd process issues the commands in the order in which they appear in the configuration. For example, in the following configuration, the execution of **policy1** causes the **show interfaces** command to be issued first, followed by the **show chassis alarms** command:

```

[edit event-options policy policy1 then execute-commands]
user@host# show
commands {
 "show interfaces";
 "show chassis alarms";
}

```

You can include variables in the command to allow data from the triggering event to be automatically included in the command syntax. The eventd process replaces each variable with values contained in the event that triggers the policy. You can use command variables of the following forms:

- **{*\$\$*.attribute-name}**—The double dollar sign (*\$\$*) notation represents the event that is triggering a policy. When combined with an attribute name, the variable is replaced by the value of the attribute name in the triggering event. For example, **{*\$\$*.interface-name}** stands for the value of the **interface-name** attribute in the triggering event.
- **{*\$event*.attribute-name}**—The **{*\$event*.attribute-name}** notation represents the most recent event that matches the specified event. The variable is replaced by the



value of the attribute name of the most recent event that matches *event*. For example, when a policy issues the `show interfaces` `{%COSD_CHAS_SCHED_MAP_INVALID.interface-name}` command, the `{%COSD_CHAS_SCHED_MAP_INVALID.interface-name}` variable is substituted by the `interface-name` attribute of the most recent `COSD_CHAS_SCHED_MAP_INVALID` event cached by the event process.

For a given event, you can view a list of event attributes that you can reference in an operational mode command by issuing the `help syslog event-name` command:

```
user@host> help syslog event-name
```

For example, in the following command output, text in angle brackets (< >) shows that `classifier-type` is an attribute of the `cosd_unknown_classifier` event:

```
user@host> help syslog cosd_unknown_classifier
Name: COSD_UNKNOWN_CLASSIFIER
Message: rtsock classifier type <classifier-type> is invalid
...
```

You can filter the output of a search by using the pipe (|) symbol. The following example lists the filters that can be used with the pipe symbol:

```
user@host# help syslog | ?
Possible completions:
count Count occurrences
display Show additional kinds of information
except Show only text that does not match a pattern
find Search for first occurrence of pattern
hold Hold text without exiting the --More-- prompt
last Display end of output only
match Show only text that matches a pattern
no-more Don't paginate output
request Make system-level requests
resolve Resolve IP addresses
save Save output text to file
trim Trim specified number of columns from start of line
```

For more information about using the pipe symbol, see the *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 by Event Policies” on page 353.

For a configuration example, see “Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval” on page 371.

## 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 {
 argument-name argument-value;
 }
 output-filename filename;
 output-format (text | xml);
 destination destination-name;
 }
 }
}
```

In the **events** statement, you can list multiple events. If one or more of the listed events occurs, the event script is executed. To view a list of the events that can be referenced in an event policy, issue the **set event-options policy *policy-name* events ?** configuration mode command:

```
[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
<event>
[Open a set of values
acct_accounting_ferror
acct_accounting_fopen_error
...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the **set event-options policy *policy-name* events ?** command. For information about referencing these system log messages in your event policies, see “Using Nonstandard System Log Messages to Trigger Event Policies” on page 352.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events to Trigger Event Policies” on page 351.

In the **event-script** statement, you can specify a script to be executed on receipt of an event. The eventd process runs the scripts in the order in which they appear in the configuration. The scripts that you reference in the **event-script** statement must be located in the `/var/db/scripts/event` directory on the router’s hard drive or the `/config/scripts/event` directory on the flash drive. For more information on event script file location, see “How Event Scripts Work” on page 413. 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 424 and “Event Policy Examples” on page 371.

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 by Event Policies” on page 353.

For the **output-filename** and **destination** statements, there are four configuration scenarios:

- You can omit the **output-filename** and **destination** statements. This option makes sense when the event script has no output. For example, the event script might execute only **request** commands, which have no output.
- You can include the **destination** statement in the configuration. You omit the **output-filename** statement in the configuration and specify an output filename in the event script instead. The script output is sent to the destination specified in the configuration. If you do not include the **destination** statement in the configuration, the script output is not uploaded.

In this scenario, the event policy extracts the filename from the event script. The event script writes the output filename as **STDOUT**. The XML syntax to use in the event script is:

```
<output>
 <event-script-output-filename>filename</event-script-output-filename>
</output>
```

The **<event-script-output-filename>** element must be the first child tag within the **<output>** parent tag.

On a router named **router2**, configure an event script action with a destination **host**, and omit the **output-filename** statement. Define the destination **host** as **ftp://user@router1/tmp**.

In the **script1.xml** 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.xml {
 destination host;
 }
 }
}
destinations {
 host {
 archive-sites {
 "ftp://user@router1/tmp" password "9XkJNbYg4ZDH.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`.

- You can include the `output-filename` and `destination` statements. If you include the `output-filename` statement in the configuration, you must also include the `destination` statement in the configuration. In this case, the script output is redirected to the output filename specified in the configuration and is sent to the destination specified in the configuration.
- You can include the `output-filename` and `destination` statements, and also specify an output filename directly within the event script. If you do this, the output filename specified in the configuration overrides the output filename specified in the event script.

## Configuring Event Policies to Ignore an Event

You can modify a policy to cause particular events to be ignored or to cause all events to be ignored during a particular time interval, to allow for maintenance for example. To configure such a policy, include the following statements at the `[edit event-options]` hierarchy level:

```
[edit event-options]
policy policy-name {
 events [events];
 then {
 ignore;
 }
}
```

In the `events` statement, you can list multiple events. To view a list of the events that can be referenced in an event policy, issue the `set event-options policy policy-name events ?` configuration mode command:

```
[edit]
user@host# set event-options policy policy-name events ?
Possible completions:
<event>
[Open a set of values
acct_accounting_ferror
acct_accounting_fopen_error
...
```

Some of the system log messages that you can reference in an event policy are not listed in the output of the `set event-options policy policy-name events ?` command. For information about referencing these system log messages in your event policies, see “Using Nonstandard System Log Messages to Trigger Event Policies” on page 352.

In addition, you can reference internally generated events, which are discussed in “Generating Internal Events to Trigger Event Policies” on page 351.

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 “Example: Ignoring Events Based on Receipt of Other Events” on page 378.

## Changing the User Privilege Level for an Event Policy Action

---

Only superusers can configure event policies. Event policy actions—such as executing event scripts, uploading files, and executing operational mode commands—are by default executed by user **root**, because the event process (eventd) runs with **root** privileges.

In some cases, you might want an event policy action to be executed with restricted privileges. For example, suppose you configure an event policy that executes a script if an interface goes down. The script includes remote procedure calls (RPCs) to change the 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* then event-script *script-name*]
- [edit event-options policy *policy-name* then execute-commands]
- [edit event-options policy *policy-name* then upload filename (*filename* | committed) 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 “Example: Associating an Optional User with an Event Policy Action” on page 374.

## Configuring Event Policies to Raise SNMP Traps

---

SNMP *traps* enable an agent to notify a network management system (NMS) of significant events by way of an unsolicited SNMP message. You can configure an



event policy action that raises traps for events based on system log messages. 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]
events [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 “Example: Raising an SNMP Trap in Response to an Event” on page 381.

## Tracing Event Policy Processing

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

By default, no events are traced. If you include the `traceoptions` statement at the `[edit event-options]` hierarchy level, the default tracing behavior is the following:

- Important events are logged in a file called `eventd` located in the `/var/log` directory.
- When the file `eventd` reaches 128 kilobytes (KB), it is renamed and compressed to `eventd.0.gz`, then `eventd.1.gz`, and so on, until there are three trace files. Then the oldest trace file (`eventd.2.gz`) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)
- Log files can be accessed only by the user who configures the tracing operation.

You cannot change the directory (`/var/log`) to which trace files are written. However, you can customize the other trace file settings by including the following statements at the `[edit event-options traceoptions]` hierarchy level:

```
[edit event-options traceoptions]
file <filename> <files number> <match regular-expression> <size size> <world-readable |
 no-world-readable>;
flag all;
flag configuration;
flag database;
flag events;
flag policy;
flag server;
flag syslog;
flag timer-events;
no-remote-trace;
```

These statements are described in the following sections:

- Configuring the Event Policy Log Filename on page 368
- Configuring the Number and Size of Event Policy Log Files on page 368
- Configuring Access to the Log File on page 368
- Configuring a Regular Expression for Lines to Be Logged on page 369
- Configuring the Trace Operations on page 369

### Configuring the Event Policy Log Filename

By default, the name of the file that records trace output is `eventd`. You can specify a different name by including the `file` statement at the `[edit event-options traceoptions]` hierarchy level:

```
[edit event-options traceoptions]
file filename;
```

### Configuring the Number and Size of Event Policy Log Files

By default, when the trace file reaches 128 kilobytes (KB) in size, it is renamed `filename.0`, then `filename.1`, and so on, until there are three trace files. Then the oldest trace file (`filename.2`) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the `[edit event-options traceoptions file <filename>]` hierarchy level:

```
[edit event-options traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 2 MB and the maximum number of files to 20. When the file that receives the output of the tracing operation (`filename`) reaches 2 MB, `filename` is renamed and compressed to `filename.0.gz` and a new file called `filename` is created.

When the new `filename` reaches 2 MB, `filename.0.gz` is renamed `filename.1.gz` and `filename` is renamed and compressed to `filename.0.gz`. This process repeats until there are 20 trace files. Then the oldest file (`filename.19.gz`) is overwritten.

The number of files can 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 `world-readable` statement at the `[edit event-options traceoptions file <filename>]` hierarchy level:

```
[edit event-options traceoptions file <filename>]
world-readable;
```

To explicitly set the default behavior, include the `no-world-readable` statement at the `[edit event-options traceoptions file <filename>]` hierarchy level:

```
[edit event-options traceoptions file <filename>]
no-world-readable;
```

## Configuring a Regular Expression for Lines to Be Logged

By default, the trace operation output includes all lines relevant to the logged events.

You can refine the output by including the `match` statement at the `[edit event-options traceoptions file <filename>]` hierarchy level and specifying a regular expression to be matched:

```
[edit event-options traceoptions file <filename>]
match regular-expression;
```

## Configuring the Trace Operations

By default, only important events are logged. You can configure the trace operations to be logged by including the following statements at the `[edit event-options traceoptions]` hierarchy level:

```
[edit event-options traceoptions]
flag all;
flag configuration;
flag database;
flag events;
flag policy;
flag server;
flag syslog;
flag timer-events;
```

Table 19 on page 369 describes the meaning of the event policy tracing flags.

**Table 19: Event Policy Tracing Flags**

| Flag          | Description                                                                            | Default Setting |
|---------------|----------------------------------------------------------------------------------------|-----------------|
| all           | Trace all operations.                                                                  | Off             |
| configuration | Log reading of configuration at the <code>[edit event-options]</code> 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             |

**Table 19: Event Policy Tracing Flags** *(continued)*

| Flag         | Description                      | Default Setting |
|--------------|----------------------------------|-----------------|
| 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 24

# Event Policy Examples

This chapter includes the following examples:

- Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval on page 371
- Examples: Assigning a Transfer Delay to an Event Policy Action on page 372
- Example: Representing the Correlating Event in an Event Policy on page 374
- Example: Associating an Optional User with an Event Policy Action on page 374
- Examples: Retrying the File Upload Action on page 375
- Examples: Triggering a Policy Based on Event Count on page 377
- Example: Ignoring Events Based on Receipt of Other Events on page 378
- Example: Correlating Events Based on Event Attributes on page 379
- Controlling Event Policy Using a Regular Expression on page 380
- Example: Generating an Internal Event Every Hour on page 380
- Example: Generating an Internal Event at Midnight on page 381
- Example: Raising an SNMP Trap in Response to an Event on page 381
- Example: Using Nonstandard System Log Messages to Trigger an Event Policy on page 381

### Example: Correlating Events Based on Receipt of Other Events Within a Specified Time Interval

---

In the following policy, a set of commands is issued and the output is logged and saved to a given location. The policy is executed if *event3*, *event4*, or *event5* occurs within 60 seconds after *event1* or *event2* occurs. The pseudocode for the policy is as follows:

```
if this event is (event3 or event4 or event5)
 and
 (event1 or event2 has been received within the last 60 seconds)
then {
 run a set of commands;
 log the output of these commands to a location;
}
```

Specify two archive sites in the configuration. The router attempts to transfer to the first archive site in the list, moving to the next site only if the transfer fails.

```
[edit event-options]
policy 1 {
 events [event3 event4 event5];
 within 60 events [event1 event2];
 then {
 execute-commands {
 commands {
 "command";
 }
 output-filename my_cmd_out;
 destination policy-1-command-dest;
 }
 }
}
destinations {
 policy-1-command-dest {
 archive-sites {
 http://robot@my.big.com/a/b;
 http://robot@my.little.com/a/b;
 }
 }
}
```

## Examples: Assigning a Transfer Delay to an Event Policy Action

This section discusses three examples.

**Example 1** Configure two event policies, `policy1` and `policy2`. The `policy1` event policy has a 5-second transfer-delay when uploading the `process.core` file to the `some-dest` destination. The `policy2` event policy has no transfer delay when uploading the `process.core` file to the same destination.

```
[edit event-options]
policy policy1 {
 events e1;
 then {
 upload filename process.core destination some-dest {
 transfer-delay 5;
 }
 }
}
policy policy2 {
 events e2;
 then {
 upload filename process.core destination some-dest;
 }
}
destinations {
 some-dest {
 archive-sites {
 "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 `process.core` file to the destination. The `policy2` event policy has a 2-second transfer delay when uploading the `process.core` file to the destination.

```

[edit event-options]
policy policy1 {
 events e1;
 then {
 upload filename process.core destination some-dest {
 transfer-delay 5;
 }
 }
}
policy policy2 {
 events e2;
 then {
 upload filename process.core destination some-dest;
 }
}
destinations {
 some-dest {
 transfer-delay 2;
 archive-sites {
 "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 `process.core` file after a transfer delay of 7 seconds (5 seconds + 2 seconds). The `policy2` event policy is executed with `root` privileges and uploads the `process.core` file after a transfer delay of 6 seconds (4 seconds + 2 seconds).

```

[edit event-options]
policy policy1 {
 events e1;
 then {
 upload filename process.core destination some-dest {
 transfer-delay 5;
 user-name user1;
 }
 }
}
policy policy2 {
 events e2;
 then {
 upload filename process.core destination some-dest {
 transfer-delay 4;
 }
 }
}
destinations {

```

```

some-dest {
 transfer-delay 2;
 archive-sites {
 "http://robot@my.little.com/foo/moo" password "password";
 "http://robot@my.big.com/foo/moo" password "password";
 }
}

```

### Example: Representing the Correlating Event in an Event Policy

---

```

[edit event-options]
policy p1 {
 events [e1 e2 e3];
 within 60 events [e4 e5 e6];
 then {
 execute-commands {
 commands {
 "show interfaces {${$.interface-name}";
 "show interfaces {$e4.interface-name}";
 "show interfaces {$*.interface-name}";
 }
 output-filename command-output.txt;
 destination some-dest;
 }
 }
}

```

In the `show interfaces {${$.interface-name}}` command, the value of the `interface-name` attribute of event `e1`, `e2`, or `e3` is substituted for the `{${$.interface-name}}` variable.

In the `show interfaces {$e4.interface-name}` command, the value of the `interface-name` attribute of the most recent `e4` event is substituted for the `{$e4.interface-name}` variable.

In the `show interfaces {$*.interface-name}` command, the value of the `interface-name` attribute of the most recent `e4`, `e5`, or `e6` event is substituted for the `{$*.interface-name}` variable. If one of `e4`, `e5`, or `e6` occurs within 60 seconds of `e1`, `e2`, or `e3`, the value of the `interface-name` attribute for that correlating event (`e4`, `e5`, or `e6`) is substituted for the `{$*.interface-name}` variable. If the correlating event does not have an `interface-name` attribute, the software does not execute the `show interfaces {$*.interface-name}` command.

If both `e4` and `e5` occur within 60 seconds of `e1`, then the value of the `interface-name` attribute for `e4` is substituted for the `{$*.interface-name}` variable. This is because the event process (eventd) searches for correlating events in sequential order as configured in the `within` statement. In this case, the order is `e4 > e5 > e6`.

### Example: Associating an Optional User with an Event Policy Action

---

Configure two event policies, `policy1` and `policy2`.



In `policy1`, associate user `user1` with the `execute-commands` action. The `execute-commands` action is executed with `user1` privileges.

In `policy2`, do not explicitly associate a user with the `event-script` action. The `event-script` action is executed with `root` privileges.

```
[edit system]
login {
 user user1 {
 class operator;
 }
}
[edit event-options]
policy p1 {
 events e1;
 then {
 execute-commands {
 commands {
 "show version";
 }
 user-name user1;
 output-filename command-output.txt;
 destination some-dest;
 }
 }
}
policy p2 {
 events e2;
 then {
 event-script script.xml {
 output-filename event-script-output.txt;
 destination some-dest;
 }
 }
}
```

## Examples: 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.

## Examples: Triggering a Policy Based on Event Count

This section discusses two examples.



**NOTE:** The RADIUS\_LOGIN\_FAIL, TELNET\_LOGIN\_FAIL, and SSH\_LOGIN\_FAIL events are not actual JUNOS 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.xml` event script, which disables the user account.

```
[edit event-options]
policy login {
 events [RADIUS_LOGIN_FAIL TELNET_LOGIN_FAIL SSH_LOGIN_FAIL];
 within 120 {
 trigger after 4;
 }
 then {
 event-script login-fail.xml {
 destination some-dest;
 }
 }
}
```

Table 20 on page 377 shows how events add to the count.

**Table 20: 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 20 on page 377 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.xml** 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.xml** event script, which disables the **roger** user account.

```
[edit event-options]
policy p2 {
 events [RADIUS_LOGIN_FAIL TELNET_LOGIN_FAIL SSH_LOGIN_FAIL];
 within 120 {
 trigger after 4;
 }
 attributes-match {
 RADIUS_LOGIN_FAIL.username matches roger;
 TELNET_LOGIN_FAIL.username matches roger;
 }
 then {
 event-script login-fail.xml {
 destination some-dest;
 }
 }
}
```

## Example: Ignoring Events Based on Receipt of Other Events

---

In the following policy, if any of *event1*, *event2*, or *event3* has occurred, and either *event4* or *event5* has occurred within the last 600 seconds, and *event6* has not occurred within the last 800 seconds, then the event that triggered the policy (*event1*, *event2*, or *event3*) is ignored, meaning system log messages are not created.

```
[edit event-options]
policy 1 {
 events [event1 event2 event3];
 within 600 events [event4 event5];
 within 800 not events event6;
 then {
 ignore;
 }
}
```

Sometimes events are generated repeatedly within a short period of time. In this case, it is redundant to execute a policy multiple times, once for each instance of the event. Event dampening allows you to slow down the execution of policies by ignoring instances of an event that occur within a specified time after another instance of the same event.

In the following example, an action is taken only if the eventd process has not received another instance of the event within the past 60 seconds. If an instance of the event has been received within the last 5 seconds, the policy is not executed and a system log message for the event is not created again.

```
[edit event-options]
policy dampen-policy {
 events event1;
 within 60 events event1;
 then {
 ignore;
 }
}
policy policy {
 events event1;
 then {
 ... actions ...
 }
}
```

## Example: Correlating Events Based on Event Attributes

---

In the following policy, the two events are correlated only if two of their parameter values match. Matching on attributes of both events ensures that the two events are related. In this case, the interface addresses must match and the physical interface (ifd) names must match.

The RPD\_KRT\_IFDCHANGE error occurs when the routing protocol process (rpd) sends a request to the kernel to change the state of an interface and the request fails. The RPD\_RDISC\_NOMULTI error occurs when an interface is configured for router discovery but the interface does not support IP multicast operations as required.

In this example, RPD\_RDISC\_NOMULTI.interface-name might be so-0/0/0.0, and RPD\_KRT\_IFDCHANGE.ifd-index might be so-0/0/0.

```
[edit event-options]
policy 1 {
 events rpd_rdisc_nomulti;
 within 500 events rpd_krt_ifdchange;
 attributes-match {
 rpd_rdisc_nomulti.interface-address equals rpd_krt_ifdchange.address;
 rpd_rdisc_nomulti.interface-name starts-with rpd_krt_ifdchange.ifd-index;
 }
 then {
 ... actions ...
 }
}
```

## Controlling Event Policy Using a Regular Expression

The following policy is executed only if the `interface-name` attribute in both traps (SNMP\_TRAP\_LINK\_DOWN and SNMP\_TRAP\_LINK\_UP) match each other and the `interface-name` attribute in the SNMP\_TRAP\_LINK\_DOWN trap starts with letter *t*. This means the policy is executed only for T1 (t1-) and T3 (t3-) interfaces. The policy is not executed when the `eventd` process receives traps from other interfaces.



**NOTE:** In system log files, the message tags appear in all uppercase letters. In the command-line interface (CLI), the message tags appear in all lowercase letters.

```
[edit event-options]
policy pol6 {
 events snmp_trap_link_down;
 within 120 events snmp_trap_link_up;
 attributes-match {
 snmp_trap_link_up.interface-name equals snmp_trap_link_down.interface-name;
 snmp_trap_link_down.interface-name matches "^t";
 }
 then {
 execute-commands {
 commands {
 "show interfaces {${$.interface-name}";
 "show configuration interfaces {${$.interface-name}";
 }
 output-filename config.txt;
 destination bsd2;
 output-format text;
 }
 }
}
```

## Example: 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.

```
[edit event-options]
generate-event every-one-hour time-interval 3600;
policy check-heartbeat {
 events every-one-hour;
 within 3601 not events every-one-hour;
 then {
 ... actions ...
 }
}
```

### Example: 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.

```
[edit event-options]
generate-event it-is-midnight time-of-day 00:00:00;
policy midnight-chores {
 events it-is-midnight;
 then {
 ... actions ...
 }
}
```

### Example: Raising an SNMP Trap in Response to an Event

---

Raise a trap and execute an associated event script in response to an event:

```
[edit event-options]
policy p1 {
 events ui_mgd_terminate;
 then {
 raise-trap;
 event-script bgp.xml {
 arguments {
 destination {ui_mgd_terminate.destination};
 code 2;
 }
 output-filename bgp-out;
 destination bsd3;
 }
 }
}
```

### Example: Using Nonstandard System Log Messages to Trigger an Event Policy

---

Reference a KERNEL system log message in an event policy. The `raise-trap` action in the `then` statement is executed only if a KERNEL event containing a message that matches "exited on signal 11" occurs.

```
[edit event-options]
policy kernel-policy {
 events KERNEL;
 attributes-match {
 KERNEL.message matches "exited on signal 11";
 }
 then {
 raise-trap;
 }
}
```





## Chapter 25

# Summary of Event Policy Configuration Statements

This chapter describes each configuration statement for event policies. The statements are organized alphabetically.

### archive-sites

---

**Syntax**    `archive-sites {  
                  url <password password>;  
                  }`

**Hierarchy Level**    [edit event-options destinations *destination-name*]

**Release Information**    Statement introduced in JUNOS 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.

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

**Related Topics**    ■    Defining Destinations for File Archiving by Event Policies on page 353

## arguments

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | arguments {<br><i>argument-name argument-value</i> ;<br>}                                                                   |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ]                                           |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.6.                                                                                  |
| <b>Description</b>              | Define command-line arguments for an event script that is invoked from an event policy.                                     |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Executing Event Scripts in an Event Policy on page 360                                                                    |

## attributes-match

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | attributes-match {<br><i>event1.attribute-name equals event2.attribute-name</i> ;<br><i>event.attribute-name matches regular-expression</i> ;<br><i>event1.attribute-name starts-with event2.attribute-name</i> ;<br>}                                                                                                                                                                                                                                                                                                                                                           |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>              | <p>Execute the policy only if the attributes of two events are correlated or if the attribute of one event matches a regular expression.</p> <p>If the <b>attributes-match</b> statement includes the <b>equals</b> or <b>starts-with</b> options, or if it includes a <b>matches</b> option that includes a clause for an event that is not specified at the [edit event-options policy <i>policy-name</i> events] hierarchy level, you must include one or more <b>within</b> statements in the same policy configuration.</p> <p>The statements are explained separately.</p> |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Related Topics</b>           | ■ Using Correlated Events to Trigger an Event Policy on page 347                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## commands

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <pre>commands {     "command"; }</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then execute-commands]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>              | Specify an operational mode command to be issued on receipt of an event.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Options</b>                  | <p><b>command</b>—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.</p> <p>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:</p> <ul style="list-style-type: none"> <li>■ <b>{<i>\$\$</i>.attribute-name}</b>—The double dollar sign (<i>\$\$</i>) 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.</li> <li>■ <b>{<i>\$event</i>.attribute-name}</b>—The dollar sign with the event name (<i>\$event</i>) 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 <i>event</i>.</li> <li>■ <b>{<i>\$*</i>.attribute-name}</b>—The dollar sign with the asterisk (<i>\$*</i>) 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.</li> </ul> |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Configuring an Event Policy to Execute Operational Mode Commands on page 357</li> <li>■ Representing the Correlating Event in an Event Policy on page 349</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## destination

---

|                                 |                                                                                                                                                                                                    |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | destination <i>destination-name</i> {<br>retry-count <i>count</i> retry-interval <i>seconds</i> ;<br>transfer-delay <i>seconds</i> ;<br>}                                                          |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ],<br>[edit event-options policy <i>policy-name</i> then execute-commands]                                         |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.<br>Support extended to the [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ] hierarchy level in JUNOS Release 7.6.      |
| <b>Description</b>              | Assign a location to which to upload command or script output for the specified policy.                                                                                                            |
| <b>Options</b>                  | <i>destination-name</i> —Name of a destination defined in the <b>destinations</b> statement at the [edit event-options] hierarchy level.<br><br>The remaining statements are defined separately.   |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                        |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Configuring an Event Policy to Execute Operational Mode Commands on page 357</li> <li>■ Executing Event Scripts in an Event Policy on page 360</li> </ul> |

## destinations

---

|                                 |                                                                                                                                                                                                                     |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <pre> destinations {   destination-name {     archive-sites {       url &lt;password password&gt;;     }     transfer-delay seconds;   } } </pre>                                                                   |
| <b>Hierarchy Level</b>          | [edit event-options]                                                                                                                                                                                                |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                          |
| <b>Description</b>              | Define one or more destinations, each with a unique name and other attributes. You can use the destination as a storage location for command output and for various files, such as system log files and core files. |
| <b>Options</b>                  | <p><i>destination-name</i>—Name of a destination.</p> <p>The remaining statements are explained separately.</p>                                                                                                     |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                              |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Defining Destinations for File Archiving by Event Policies on page 353</li> </ul>                                                                                          |

## equals

---

|                                 |                                                                                                                                        |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <i>event1.attribute-name equals event2.attribute-name;</i>                                                                             |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> attributes-match]                                                                        |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                             |
| <b>Description</b>              | Execute the policy only if the specified attribute of <i>event1</i> equals the specified attribute of <i>event2</i> .                  |
| <b>Options</b>                  | <p><i>event1.attribute-name</i>—Attribute of one event.</p> <p><i>event2.attribute-name</i>—Attribute of another event.</p>            |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p> |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Using Correlated Events to Trigger an Event Policy on page 347</li> </ul>                     |

## event-options

---

```

Syntax event-options {
 destinations {
 destination-name {
 archive-sites {
 url <password password>;
 }
 transfer-delay seconds;
 }
 }
 event-script {
 file filename {
 refresh;
 refresh-from url;
 remote-execution {
 remote-hostname {
 passphrase user-password;
 username user-login;
 }
 }
 source url;
 }
 }
 refresh;
 refresh-from url;
 traceoptions {
 file <filename> <files number> <size size> <world-readable | no-world-readable>;
 flag flag;
 no-remote-trace;
 }
 }
 generate-event event-name {
 time-interval seconds;
 time-of-day hh:mm:ss;
 }
 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 {
 argument-name argument-value;
 }
 output-filename filename;
 destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 }
 }
 }
 }

```

```

 }
 execute-commands {
 commands {
 "command";
 }
 destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 user-name username;
 }
 ignore;
 raise-trap;
 upload filename (filename | committed) destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 user-name username;
 }
}

}

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

```

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Hierarchy Level</b>          | [edit]                                                                                                                      |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Configure event policies.<br><br>The statements are explained separately.                                                   |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |

**Related Topics** ■ [Configuring Event Policy on page 345](#)

## event-script

---

**Syntax** `event-script filename {  
     arguments {  
         argument-name argument-value;  
     }  
     destination destination-name {  
         retry-count count retry-interval seconds;  
         transfer-delay seconds;  
     }  
     output-filename filename;  
     output-format (text | xml);  
     user-name username;  
}`

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in JUNOS 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.

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

**Related Topics** ■ [Executing Event Scripts in an Event Policy on page 360](#)



## events

---

See the following sections:

- events (Associating Events with a Policy) on page 392
- events (Correlating Events with Each Other) on page 392

**events (Associating Events with a Policy)**

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | events [ events ];                                                                                                          |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> ]                                                                             |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Create a list of events that trigger this policy. If one or more of the listed events occurs, the policy is executed.       |
| <b>Options</b>                  | [ events ]—List of events. Events can be internally generated, or they can be generated by JUNOS Software processes.        |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Using Correlated Events to Trigger an Event Policy on page 347                                                            |

**events (Correlating Events with Each Other)**

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | events [ events ];                                                                                                          |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> within <i>seconds</i> ]                                                       |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Create a list of events that must occur within a specified time interval for the policy to be triggered.                    |
| <b>Options</b>                  | [ events ]—List of events. Events can be internally generated, or they can be generated by JUNOS Software processes.        |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Using Correlated Events to Trigger an Event Policy on page 347                                                            |

## execute-commands

---

**Syntax**

```
execute-commands {
 commands {
 "command";
 }
 destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 user-name username;
}
```

**Hierarchy Level** [edit event-options policy *policy-name* then]

**Release Information** Statement introduced in JUNOS 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.

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

**Related Topics** ■ Configuring an Event Policy to Execute Operational Mode Commands on page 357

## generate-event

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | generate-event <i>event-name</i> {<br>time-interval <i>seconds</i> ;<br>time-of-day <i>hh:mm:ss</i> ;<br>}                  |
| <b>Hierarchy Level</b>          | [edit event-options]                                                                                                        |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Generate an internal event, based on a time interval or the time of day.                                                    |
| <b>Options</b>                  | <i>event-name</i> —Name of an internally generated event.<br><br>The statements are explained separately.                   |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Generating Internal Events to Trigger Event Policies on page 351                                                          |

## ignore

---

|                                 |                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | ignore;                                                                                                                                                                                                                                                                                                                                       |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then]                                                                                                                                                                                                                                                                                           |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>              | 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 <b>ignore</b> statement in a policy configuration, you cannot configure any other actions in the policy. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                                                                   |
| <b>Related Topics</b>           | ■ Configuring Event Policies to Ignore an Event on page 365                                                                                                                                                                                                                                                                                   |

## matches

---

|                                 |                                                                                                                                                                                                                       |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <i>event.attribute-name</i> matches <i>regular-expression</i> ;                                                                                                                                                       |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> attributes-match]                                                                                                                                                       |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                            |
| <b>Description</b>              | Execute the policy only if the specified attribute of <i>event</i> matches a regular expression.                                                                                                                      |
| <b>Options</b>                  | <p><i>event.attribute-name</i>—Event attribute to compare to a regular expression.</p> <p><i>regular-expression</i>—Regular expression to compare.</p>                                                                |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Using Correlated Events to Trigger an Event Policy on page 347</li> <li>■ Using Regular Expressions to Refine the Set of Events That Trigger a Policy on page 350</li> </ul> |

## not

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | not events [ <i>events</i> ];                                                                                               |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> within <i>seconds</i> ]                                                       |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Create a list of events that must not occur within the specified time interval for the policy to be triggered.              |
| <b>Options</b>                  | [ <i>events</i> ]—List of events. Events can be internally generated, or they can be generated by JUNOS Software processes. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Using Correlated Events to Trigger an Event Policy on page 347                                                            |

## output-filename

---

|                                 |                                                                                                                                                                                                    |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | output-filename <i>filename</i> ;                                                                                                                                                                  |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ],<br>[edit event-options policy <i>policy-name</i> then execute-commands]                                         |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.<br>Support at the [edit event-options policy <i>policy-name</i> then event-script <i>script-name</i> ] hierarchy level introduced in JUNOS Release 7.6. |
| <b>Description</b>              | Assign a filename to which to write command or script output for the specified commands or script. For op scripts, this statement is optional.                                                     |
| <b>Options</b>                  | <i>filename</i> —Name of a file in which to write command or script output.                                                                                                                        |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                        |
| <b>Related Topics</b>           | ■ Configuring an Event Policy to Execute Operational Mode Commands on page 357<br>■ Executing Event Scripts in an Event Policy on page 360                                                         |

## output-format

---

|                                 |                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | output-format (text   xml);                                                                                                                                                                                                                                                                                                         |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ],<br>[edit event-options policy <i>policy-name</i> then execute-commands]                                                                                                                                                                          |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.<br>Support at the [edit event-options policy <i>policy-name</i> then event-script <i>script-name</i> ] hierarchy level introduced in JUNOS Release 8.3.                                                                                                                                  |
| <b>Description</b>              | Specify the format (ASCII text or XML) for the output of the specified commands or script.                                                                                                                                                                                                                                          |
| <b>Options</b>                  | <p>text—Formatted ASCII text.</p> <p>xml—JUNOS Extensible Markup Language (XML) tags.<br/> <b>Default:</b> xml at the [edit event-options policy <i>policy-name</i> then execute-commands] hierarchy level and text at the [edit event-options policy <i>policy-name</i> then event-script <i>script-name</i>] hierarchy level.</p> |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                                                                                                                              |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Configuring an Event Policy to Execute Operational Mode Commands on page 357</li> <li>■ Executing Event Scripts in an Event Policy on page 360</li> </ul>                                                                                                                                  |

## policy

---

**Syntax** `policy policy-name {`  
     `attributes-match {`  
         `event1.attribute-name equals event2.attribute-name;`  
         `event.attribute-name matches regular-expression;`  
         `event1.attribute-name starts-with event2.attribute-name;`  
     `}`  
     `events [ events ];`  
     `then {`  
         ... *the then subhierarchy appears at the end of the [edit event-options policy*  
             *policy-name] hierarchy level ...*  
     `}`  
     `within seconds {`  
         `events [ events ];`  
         `not events [ events ];`  
         `trigger (on | after | until) event-count;`  
     `}`  
  
     `then {`  
         `event-script filename {`  
             `arguments {`  
                 `argument-name argument-value;`  
             `}`  
             `destination destination-name {`  
                 `retry-count count retry-interval seconds;`  
                 `transfer-delay seconds;`  
             `}`  
             `output-filename filename;`  
             `output-format (text | xml);`  
             `user-name username;`  
         `}`  
         `execute-commands {`  
             `commands {`  
                 `"command";`  
             `}`  
             `destination destination-name {`  
                 `retry-count count retry-interval seconds;`  
                 `transfer-delay seconds;`  
             `}`  
             `output-filename filename;`  
             `output-format (text | xml);`  
             `user-name username;`  
         `}`  
         `ignore;`  
         `raise-trap;`  
         `upload filename (filename | committed) destination destination-name {`  
             `retry-count count retry-interval seconds;`  
             `transfer-delay seconds;`  
             `user-name username;`  
         `}`  
     `}`  
     `}`  
     `}`



|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Hierarchy Level</b>          | [edit event-options]                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>              | <p>Define an event policy to be processed by the eventd process. If you configure a policy, the <b>events</b> and <b>then</b> statements are mandatory.</p> <p>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 <b>ignore</b> statement are executed.</p> |
| <b>Default</b>                  | If you do not configure a policy for an event, the event is recorded in the system log.                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Options</b>                  | <p><i>policy-name</i>—Name of an event policy.</p> <p>The statements are explained separately.</p>                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                                                                                                                                                                                                                                                                                          |

**Related Topics** ■ [Configuring Event Policy on page 345](#)

## raise-trap

---

|                                 |                                                                                                                                                                                                                                                                                 |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | raise-trap;                                                                                                                                                                                                                                                                     |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then]                                                                                                                                                                                                                             |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 8.1.                                                                                                                                                                                                                                      |
| <b>Description</b>              | Define a policy that raises an SNMP trap in response to an event. If one or more of the listed events occur, the system log message for the event is converted into a trap. This enables an agent to notify a trap-based network management system (NMS) of significant events. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                     |
| <b>Related Topics</b>           | ■ <a href="#">Configuring Event Policies to Raise SNMP Traps on page 366</a>                                                                                                                                                                                                    |

## retry-count

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | retry-count <i>number</i> retry-interval <i>seconds</i> ;                                                                                                                                                                                                                                                                                                              |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> destination <i>destination-name</i> ],<br>[edit event-options policy <i>policy-name</i> then event-script destination <i>destination-name</i> ],<br>[edit event-options policy <i>policy-name</i> then upload filename ( <i>filename</i>   committed) destination <i>destination</i> ] |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 8.4.                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>              | Configure an event policy to retry a file upload operation if the first attempt fails.                                                                                                                                                                                                                                                                                 |
| <b>Default</b>                  | If you do not include this statement, the file upload operation is attempted one time only.                                                                                                                                                                                                                                                                            |
| <b>Options</b>                  | <i>number</i> —Number of retries.<br><br><i>retry-interval seconds</i> —Length of time to wait between retries.                                                                                                                                                                                                                                                        |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                                                                                            |
| <b>Related Topics</b>           | ■ <a href="#">Configuring an Event Policy to Retry the File Upload Action on page 356</a>                                                                                                                                                                                                                                                                              |

## starts-with

---

|                                 |                                                                                                                                        |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <code>event1.attribute-name starts-with event2.attribute-name;</code>                                                                  |
| <b>Hierarchy Level</b>          | <code>[edit event-options policy <i>policy-name</i> attributes-match <i>event1.attribute-name</i>]</code>                              |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                             |
| <b>Description</b>              | Execute the policy only if the specified attribute of <i>event1</i> starts with the specified attribute of <i>event2</i> .             |
| <b>Options</b>                  | <p><i>event1.attribute-name</i>—Attribute of one event.</p> <p><i>event2.attribute-name</i>—Attribute of another event.</p>            |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p> |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Using Correlated Events to Trigger an Event Policy on page 347</li> </ul>                     |

**then**

```

Syntax then {
 event-script filename {
 arguments {
 argument-name argument-value;
 }
 destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 user-name username;
 }
 execute-commands {
 commands {
 "command";
 }
 destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 }
 output-filename filename;
 output-format (text | xml);
 user-name username;
 }
 ignore;
 raise-trap;
 upload filename (filename | committed) destination destination-name {
 retry-count count retry-interval seconds;
 transfer-delay seconds;
 user-name username;
 }
 }

```

**Hierarchy Level** [edit event-options policy *policy-name*]

**Release Information** Statement introduced in JUNOS 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.

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

- Related Topics**
- Configuring an Event Policy to Upload Files on page 354
  - Configuring an Event Policy to Execute Operational Mode Commands on page 357
  - Executing Event Scripts in an Event Policy on page 360

- Configuring Event Policies to Ignore an Event on page 365
- Configuring Event Policies to Raise SNMP Traps on page 366

## time-interval

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | time-interval <i>seconds</i> ;                                                                                              |
| <b>Hierarchy Level</b>          | [edit event-options generate-event <i>event-name</i> ]                                                                      |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Configure a frequency at which to generate a particular event.                                                              |
| <b>Options</b>                  | <i>seconds</i> —Time interval between internally generated events.<br><b>Range:</b> 60 through 604,800 seconds              |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Generating Internal Events to Trigger Event Policies on page 351                                                          |

## time-of-day

---

|                                 |                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | time-of-day <i>hh:mm:ss</i> ;                                                                                               |
| <b>Hierarchy Level</b>          | [edit event-options generate-event <i>event-name</i> ]                                                                      |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                  |
| <b>Description</b>              | Configure a time of day at which to generate a particular event.                                                            |
| <b>Options</b>                  | <i>hh:mm:ss</i> —Time of day at which to generate an event.                                                                 |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration. |
| <b>Related Topics</b>           | ■ Generating Internal Events to Trigger Event Policies on page 351                                                          |

## traceoptions

---

**Syntax**    `traceoptions {  
               file <filename> <files number> <match regular-expression> <size size> <world-readable |  
               no-world-readable>;  
               flag flag;  
               no-remote-trace;  
           }`

**Hierarchy Level**    [edit event-options]

**Release Information**    Statement introduced in JUNOS Release 7.5.

**Description**    Define tracing operations for event policies.

**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 and compressed to *trace-file.0.gz*, then *trace-file.1.gz*, 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
- `server`—Log communication with processes that are generating events
- `timer-events`—Log internally generated events

*match regular-expression*—(Optional) Refine the output to include lines that contain the regular expression.

**size** *size*—(Optional) Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named *trace-file* reaches this size, it is renamed and compressed to *trace-file.0.gz*. When the *trace-file* again reaches its maximum size, *trace-file.0.gz* is renamed *trace-file.1.gz* and *trace-file* is renamed and compressed to *trace-file.0.gz*. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option and filename.

**Syntax:** *xk* to specify KB, *xm* to specify MB, or *xg* to specify GB

**Range:** 10 KB through 1 GB

**Default:** 128 KB

**world-readable**—(Optional) Enable unrestricted file access.

**Required Privilege Level**

**maintenance**—To view this statement in the configuration.

**maintenance-control**—To add this statement to the configuration.



**Related Topics** ■ [Tracing Event Policy Processing on page 367](#)

## transfer-delay

---

**Syntax** transfer-delay *seconds*;

**Hierarchy Level** [edit event-options destinations *destination-name*],  
 [edit event-options policy *policy-name* then event-script *filename*  
   destination *destination-name*],  
 [edit event-options policy *policy-name* then execute-commands  
   destination *destination-name*],  
 [edit event-options policy *policy-name* then upload filename (*filename* | committed)  
   destination *destination*]

**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.

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

**Related Topics** ■ [Defining Destinations for File Archiving by Event Policies on page 353](#)  
 ■ [Configuring the Delay Before Files are Uploaded by an Event Policy on page 355](#)

## trigger

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | trigger (on   after   until) <i>event-count</i> ;                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> within <i>seconds</i> ]                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 8.4.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>              | Configure an event policy to be triggered if an event or set of events occurs <i>event-count</i> times within a specified time period.                                                                                                                                                                                                                                                                                                                           |
| <b>Default</b>                  | If you do not include this statement, the policy is executed on receipt of the first configured event.                                                                                                                                                                                                                                                                                                                                                           |
| <b>Options</b>                  | <p><b>after <i>event-count</i></b>—The policy is executed when the number of matching events received equals <i>number</i> + 1.</p> <p><b>on <i>event-count</i></b>—The policy is executed when the number of matching events received equals <i>number</i>.</p> <p><b>until <i>event-count</i></b>—The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals <i>number</i>.</p> |
| <b>Required Privilege Level</b> | <p>maintenance—To view this statement in the configuration.</p> <p>maintenance-control—To add this statement to the configuration.</p>                                                                                                                                                                                                                                                                                                                           |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ Triggering an Event Policy Based on Event Count on page 350</li> </ul>                                                                                                                                                                                                                                                                                                                                                  |

## upload

---

**Syntax**    upload filename (*filename* | committed) destination *destination-name* {  
                   retry-count *count* retry-interval *seconds*;  
                   transfer-delay *seconds*;  
                   user-name *username*;  
                   }

**Hierarchy Level**    [edit event-options policy *policy-name* then]

**Release Information**    Statement introduced in JUNOS Release 7.5.  
                                  committed option to filename statement introduced in JUNOS Release 8.1.

**Description**    On receipt of an event, upload the committed configuration file to a destination.

**Options**    destination *destination-name*—Name of the destination for the uploaded file. It must be defined in the **destinations** statement at the [edit event-options] hierarchy level.

filename (*filename* | committed)—Name of the file to upload. Specify either the word **committed** to upload the most recently committed configuration file, or the filename of another file.

The remaining statements are explained separately.

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

**Related Topics**    ■ destinations  
                          ■ Configuring an Event Policy to Upload Files on page 354

## user-name

---

|                                 |                                                                                                                                                                                                                                                                                                      |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <code>user-name username;</code>                                                                                                                                                                                                                                                                     |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> then event-script <i>filename</i> ],<br>[edit event-options policy <i>policy-name</i> then execute-commands],<br>[edit event-options policy <i>policy-name</i> then upload filename ( <i>filename</i>   committed)<br>destination <i>destination</i> ] |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 8.4.                                                                                                                                                                                                                                                           |
| <b>Description</b>              | Associate a user with an action in an event policy. The event policy action is executed under the privileges of the associated user.                                                                                                                                                                 |
| <b>Default</b>                  | If you do not associate a user with an action, the action is executed as user <code>root</code> .                                                                                                                                                                                                    |
| <b>Options</b>                  | <i>username</i> —A username that is configured at the [edit system login] hierarchy level.                                                                                                                                                                                                           |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                                          |
| <b>Related Topics</b>           | ■ Changing the User Privilege Level for an Event Policy Action on page 366                                                                                                                                                                                                                           |

## within

---

|                                 |                                                                                                                                                                        |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <code>within seconds {<br/>    events [ <i>events</i> ];<br/>    not events [ <i>events</i> ];<br/>    trigger (after   on   until) <i>event-count</i>;<br/>}</code>   |
| <b>Hierarchy Level</b>          | [edit event-options policy <i>policy-name</i> ]                                                                                                                        |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 7.5.                                                                                                                             |
| <b>Description</b>              | Create a list of events that must (or must not) occur within a specified time interval for the policy to be triggered.<br><br>The statements are explained separately. |
| <b>Options</b>                  | <i>seconds</i> —Interval between events.<br><b>Range:</b> 60 through 604,800 seconds                                                                                   |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                            |
| <b>Related Topics</b>           | ■ Using Correlated Events to Trigger an Event Policy on page 347                                                                                                       |

## **Part 5**

# **Event Scripts**

- Event Scripts Overview on page 413
- Writing Event Scripts on page 415
- Configuring Event Scripts on page 423
- Event Script Examples on page 435
- Summary of Event Script Configuration Statements on page 437



## Chapter 26

# Event Scripts Overview

This chapter discusses the following topics:

- Event Script Programming Overview on page 413
- How Event Scripts Work on page 413

### Event Script Programming 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. router output when triggered by previously configured event policy scripts. Event scripts are executed by the event process (eventd).

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.

Event scripts are based on two application programming interfaces (APIs) to the JUNOS Software: the JUNOS XML API and the JUNOScript API, which are discussed in “JUNOScript API and JUNOS XML API Overview” 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 “XSLT Overview” on page 15. For more information about SLAX, see “SLAX Overview” on page 27.

### How Event Scripts Work

---

Event scripts initiate operational commands when triggered by an event policy. When an event policy is triggered, this policy forwards event details to the event script. You enable event scripts by listing the names of one or more event script files within

the `[edit event-options event-script]` hierarchy level. These scripts contain instructions that execute operational mode commands and inspect the output automatically.

Event scripts are invoked within an event policy. For information about event policies, see “Event Notifications and Policies Overview” on page 341 and “Executing Event Scripts in an Event Policy” on page 360.

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.



## Chapter 27

# Writing Event Scripts

This chapter explains how to write event scripts and includes the following topics:

- Required Boilerplate for Event Scripts on page 415
- Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation on page 417
- Using RPCs and Operational Mode Commands in Event Scripts on page 417
- Capturing and Using Event Details and Remote Execution Details In Event Scripts on page 420

### Required Boilerplate for Event Scripts

---

When you write event scripts, you use Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) tools provided with the JUNOS Software. These tools include basic boilerplate that you must include in all commit scripts, optional extension functions that accomplish scripting tasks more easily, and named templates that make commit scripts easier to read and write, which you import from a file called `junos.xml`. For more information about the extension functions and templates, see “JUNOS Extension Functions Overview” on page 39.

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.

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

#### XSLT Boilerplate for Event Scripts

```
1 <?xml version="1.0" standalone="yes"?>
2 <xsl:stylesheet version="1.0"
3 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4 xmlns:junos="http://xml.juniper.net/junos/*/junos"
5 xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
6 xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0">
7 <xsl:import href="../import/junos.xml"/>
8
9 <xsl:template match="configuration">
10 <event-script-results>
11 <!-- ... Insert your code here ... -->
```

```

10 </event-script-results>
11 </xsl:template>
 <!-- ... insert additional template definitions here ... -->
12 </xsl:stylesheet>

```

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

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

Line 2 opens the style sheet and specifies the XSLT version as 1.0.

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

Lines 3 through 6 list all the namespace mappings commonly used in event scripts. Not all of these prefixes are used in this example, but it is not an error to list namespace mappings that are not referenced. Listing 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 referenced as `../import/junos.xml`, which ships as part of the JUNOS Software (in the file `/usr/libdata/cscript/import/junos.xml`). The `junos.xml` file contains a set of named templates you can call in your scripts. These named templates are discussed in “Importing the `junos.xml` File” on page 51.

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

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 Overview” 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>
```

Between Line 11 and Line 12, you can define additional XSLT templates that are called from within the `<xsl:template match="/">` template.

Line 12 closes the style sheet and the event script.

#### SLAX Boilerplate for Event Scripts

```

12 </xsl:stylesheet>

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

match / {
 <event-script-results> {
 /*
 * Insert your code here
 */
 }
}

```

## Mapping Operational Mode Commands and Output Fields to JUNOS XML Notation

In event scripts, you use tag elements from the JUNOS XML API to represent operational mode commands and output fields. For the JUNOS XML equivalent of commands and output fields, consult the *JUNOS XML API Operational Reference*.

You can also display JUNOS XML by directing the output from the `show` command to the `| display xml` command:

```
user@host> operational-mode-command | display xml
```

For example:

```

user@host> show interfaces terse | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
 <interface-information
 xmlns="http://xml.juniper.net/junos/10.0R10/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>
 ...
 </interface-information>
</rpc-reply>

```

## Using RPCs and Operational Mode Commands in Event Scripts

Most JUNOS operational mode commands have XML equivalents. These XML commands can be executed remotely using the *remote procedure call* (RPC) protocol.

All operational mode commands that have XML equivalents are listed in the *JUNOS XML API Operational Reference*.

RPC and operational mode command use in event scripts is discussed in more detail in the following sections:

- Using RPCs in Event Scripts on page 418
- Using Operational Mode Commands in Event Scripts on page 419

## Using RPCs in Event Scripts

You can invoke RPCs in event scripts. For each event script that invokes RPCs, you must include the `remote-execution` statement at the `[edit event-options event-script file filename]` hierarchy level and specify the machine where the RPC is executed as `remote-hostname`. You must also include the `username` and `passphrase` statements for each remote machine.

```
[edit event-options event-script file filename]
remote-execution {
 remote-hostname {
 username username;
 passphrase passphrase;
 }
}
```

The remote hostnames and their corresponding username and passphrase, in addition to the event details, are passed as input to the event script when it is triggered by an event policy. For more information about the details that are forwarded to the event script, see “Capturing and Using Event Details and Remote Execution Details In Event Scripts” on page 420. A connection handle to the remote host is generated with the `jcs:open()` function using `remote-hostname`, `username`, and `passphrase` as input parameters; for more information, see “`jcs:open()` Function” on page 44. The following code obtains a connection handle for each remote host included in the configuration:

|                    |                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>XSLT Syntax</b> | <pre>&lt;xsl:for-each select="event-script-input/remote-execution-details"&gt;   &lt;xsl:variable name="d" select="remote-execution-detail"/&gt;   &lt;xsl:variable name="connection"     select="jcs:open(\$d/remote-hostname,\$d/username,\$d/passphrase)"/&gt;   ... &lt;/xsl:for-each&gt;</pre> |
| <b>SLAX Syntax</b> | <pre>for-each (event-script-input/remote-execution-details) {   var \$d = remote-execution-detail;   var \$connection = jcs:open(\$d/remote-hostname,\$d/username,\$d/passphrase);   ... }</pre>                                                                                                    |

After obtaining a connection handle to the remote device, the event script can execute RPCs with the `jcs:execute()` extension function, which is described in “`jcs:execute()` Function” on page 42. To use an RPC in the event script, include the RPC in a variable declaration and execute it with the `jcs:execute()` function; the connection handle and RPC variable declaration are provided as parameters to the `jcs:execute()` function.

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>XSLT Syntax</b> | <pre>&lt;xsl:variable name="rpc"&gt;   &lt;get-interface-information/&gt; # JUNOS RPC for the show interfaces command &lt;/xsl:variable&gt; &lt;xsl:variable name="out" select="jcs:execute(\$connection, \$rpc)"/&gt;</pre> |
| <b>SLAX Syntax</b> | <pre>var \$rpc = &lt;get-interface-information&gt;; var \$out = jcs:execute(\$connection, \$rpc);</pre>                                                                                                                      |

where `$connection` is the connection handle to the remote host. Any number of RPCs can be executed within the context of this connection handle until it is closed with the `jcs:close()` function.

### Using Operational Mode Commands in Event Scripts

Some operational mode commands do not have XML equivalents. If a command is not listed in the *JUNOS XML API Operational Reference*, it does not have an XML equivalent.

Another way to determine whether a command has an XML equivalent is to issue the command followed by the `| display xml` command:

```
user@host> operational-mode-command | display xml
```

If the output includes only tag elements like `<output>`, `<cli>`, and `<banner>`, the command might not have an XML equivalent. In the following example, the output indicates that the `show host` command has no XML equivalent:

```
user@host> show host hostname | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.0R1/junos">
 <output>
 ...
 </output>
 <cli>
 <banner></banner>
 </cli>
</rpc-reply>
```



**NOTE:** For some commands that have an XML equivalent, the output of the piped `| display xml` command does not include tag elements other than `<output>`, `<cli>`, and `<banner>` only because the relevant feature is not configured. For example, the `show services cos statistics forwarding-class` command has an XML equivalent that returns output in the `<service-cos-forwarding-class-statistics>` response tag, but if the configuration does not include any statements at the `[edit class-of-service]` hierarchy level then there is no actual data for the `show services cos statistics forwarding-class | display xml` command to display. The output is something like this:

```
user@host> show services cos statistics forwarding-class | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/8.3I0/junos">
 <cli>
 <banner></banner>
 </cli>
</rpc-reply>
```

For this reason, the information in the *JUNOS XML API Operational Reference* is normally more reliable.

An event script can include commands that have no XML equivalent. Use the `<command>`, `<xsl:value-of>`, and `<output>` elements in the script, as shown in the following code snippet. This snippet is expanded and fully described in “Example: Displaying DNS Hostname Information in an Op Script” on page 313.

```
<xsl:variable name="query">
 <command>
 <xsl:value-of select="concat('show host ', $hostname)"/>
 </command>
</xsl:variable>
<xsl:variable name="result" select="jcs:invoke($query)"/>
<xsl:variable name="host" select="$result"/>
<output>
 <xsl:value-of select="concat('Name: ', $host)"/>
</output>
...
```

## Capturing and Using Event Details and Remote Execution Details In Event Scripts

When an event script is triggered by an event policy, the initiating event policy forwards a set of event details to the triggered event script. These event details can be captured, evaluated, and sent to log files as required. In addition, any configured remote execution details are also forwarded to the event script. The remote execution details allow the event script to invoke remote procedure calls as detailed in “Using RPCs and Operational Mode Commands in Event Scripts” on page 417.

Two types of event details are returned: triggered events and received events. *Triggered events* record the details of the event that triggered the policy. *Received events* record the details of events that happened before the triggering event. Event details and remote execution details are forwarded to the event script as XML in the following format:

```
<event-script-input>
 <trigger-event>
```

```

<id>event-id</id>
<type>event-type</type>
<generation-time>timestamp</generation-time>
<process>
 <name>process-name</name>
 <pid>pid</pid>
</process>
<hostname>hostname</hostname>
<facility>facility-string</facility>
<severity>severity-string</severity>
<attribute-list>
 <attribute>
 <name>attribute-name</name>
 <<value>attribute-value</value>
 </attribute>
</attribute-list>
</trigger-event>
<received-events>
 <received-event>
 <id>event-id</id>
 <type>event-type</type>
 <generation-time>timestamp</generation-time>
 <process>
 <name>process-name</name>
 <pid>pid</pid>
 </process>
 <hostname>hostname</hostname>
 <facility>facility-string</facility>
 <severity>severity-string</severity>
 <attribute-list>
 <attribute>
 <name>attribute-name</name>
 <<value>attribute-value</value>
 </attribute>
 </attribute-list>
 </received-event>
</received-events>
<remote-execution-details>
 <remote-execution-detail>
 <remote-hostname>hostname</remote-hostname>
 <username>username</username>
 <passphrase>passphrase</passphrase>
 </remote-execution-detail>
</remote-execution-details>
</event-script-input>

```

For information about one method for using event details, see “Example: Limiting Event Script Output Based on a Specific Event Type” on page 435.





## Chapter 28

# 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 event-options] hierarchy level:

```
[edit event-options]
event-script {
 file filename {
 refresh;
 refresh-from url;
 remote-execution {
 remote-hostname {
 passphrase user-password;
 username user-login;
 }
 }
 source url;
 }
 refresh;
 refresh-from url;
 traceoptions {
 file <filename> <files number> <size size> <world-readable | no-world-readable>;
 flag flag;
 no-remote-trace;
 }
}
```

This chapter discusses the following topics:

- Implementing Event Scripts on page 424
- Enabling an Event Script on page 425
- Executing an Event Script on page 426
- Storing Event Scripts in Flash Memory on page 426
- Specifying a Master Source for an Event Script on page 426
- Updating an Event Script from the Master Source on page 427

- Updating an Event Script from an Alternate Location on page 428
- Converting an Event Script from XSLT to SLAX on page 428
- Converting an Event Script from SLAX to XSLT on page 429
- Tracing Event Script Processing on page 429

## Implementing Event Scripts

---

This section provides directions for using event scripts on your router.

- Installing Event Scripts on a Router on page 424
- Replacing an Event Script on page 424

### Installing Event Scripts on a Router

To install event scripts on a router, follow these steps:

1. Write the event script.

For information on writing event scripts, see “Writing Event Scripts” on page 415. For event script examples, see “Event Script Examples” on page 435.

2. Copy the script to the `/var/db/scripts/event` directory on the hard drive or the `/config/scripts/event` directory on the flash drive; for information about setting the storage location for scripts, see “Storing Event Scripts in Flash Memory” on page 426. Only users who belong to the JUNOS **super-user** login class can access and edit files in these directories.



**NOTE:** If the router or switch has dual Routing Engines and you want to enable the event script to execute on both Routing Engines, you must copy the script to the `/var/db/scripts/event` or `/config/scripts/event` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

---

3. Enable the script by including the `file filename` statement at the `[edit event-options event-script]` hierarchy level. For instructions, see “Enabling an Event Script” on page 425.
4. Issue the `commit` command.

After the commit operation completes, the event script is loaded into memory and ready for automatic execution in response to system log events. For more information, see “Executing Event Scripts in an Event Policy” on page 360.

### Replacing an Event Script

You can update or replace an existing event script without changing the router’s configuration or disrupting operations. Follow these steps:

1. Edit or write the new event script.

For information on writing event scripts, see “Writing Event Scripts” on page 415.

2. Copy the script to the `/var/db/scripts/event` directory on the hard drive or the `/config/scripts/event` directory on the flash drive; for information about setting the storage location for scripts, see “Storing Event Scripts in Flash Memory” on page 426. Only users who belong to the JUNOS **super-user** login class can alter files in these directories.



**NOTE:** If the router or switch has dual Routing Engines, remember to copy the script to the `/var/db/scripts/event` or `/config/scripts/event` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

---

3. Issue the `request system scripts event-scripts reload` operational mode command.

```
user@host> request system scripts event-scripts reload
```

All event scripts are reloaded into the `eventd` process’ memory.

## Enabling an Event Script

---

Event scripts are stored on a router or switch’s hard drive (in the `/var/db/scripts/event` directory) or flash drive (in the `/config/scripts/event` directory). Only users in the JUNOS superuser login class can access and edit files in these directories. For information about setting the storage location for scripts, see “Storing Event Scripts in Flash Memory” on page 426.



**NOTE:** If the router or switch has dual Routing Engines and you want to enable an event script to execute on both Routing Engines, you must copy the script to the `/var/db/scripts/event` or `/config/scripts/event` directory on both Routing Engines. The `commit synchronize` command does not automatically copy scripts between Routing Engines.

---

You must enable event op script before it can be executed. Include the `file filename` statement at the `[edit event-options events-script]` hierarchy level, specifying the name of an Extensible Stylesheet Language Transformations (XSLT) or Stylesheet Language Alternative Syntax (SLAX) file containing an event script. Only users who belong to the JUNOS **super-user** login class can enable event scripts.

```
[edit event-options event-script]
file filename;
```

The filename of an event script written in SLAX must include the `.slax` extension for the script to be enabled and executed. No particular filename extension is required for event scripts written in XSLT, but we strongly recommend that you append the `.xsl` extension.

To determine which event scripts are currently active on the router, either list the contents of the `/var/run/scripts/event` directory or use the `show` command to display the files included at the `[edit event-options event-script]` hierarchy level.

## Executing an Event Script

---

Unlike commit scripts, event scripts do not execute during a commit operation. When you issue the `commit` command, event scripts enabled at the `[edit event-options event-script]` hierarchy level are placed into system memory and enabled for execution. After the commit operation completes, an event script is executed in response to an event notification within an event policy. For more information, see “Executing Event Scripts in an Event Policy” on page 360.

## Storing Event Scripts in Flash Memory

---

By default, event scripts are stored in the `/var/db/scripts/event` directory on the router's hard drive. To store them in flash memory instead, include 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; event scripts are stored in the `/config/scripts/event` directory in flash memory. Changing the scripts' physical location has no effect on their operation.



**NOTE:** When you add or remove the `load-scripts-from-flash` statement in the configuration, you must manually move scripts from the hard drive to the flash drive, or vice versa, as appropriate. They are not moved automatically.

## Specifying a Master Source for an Event Script

---

You can store a master copy of each event script in a central repository. This eases file management because you can make changes to the master event script in one place and then update the copy on each router where the event script is currently enabled.

To define the location of the master source file for an event script, include the `source` statement at the `[edit event-options event-script file filename]` hierarchy level:

```
[edit event-options event-script file filename]
source url;
```

- *filename*—Name of the event script.
- *url*—URL of the event script's master source file. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

The following example specifies an HTTP URL as the remote source for the `iso.xml` file:

```
[edit event-options event-script]
file iso.xml {
 source http://my.example.com/pub/scripts/iso.xml;
}
```

Including the `source` statement in the configuration does not affect the local copy of the event script until you issue the `set refresh` command as described in “Updating an Event Script from the Master Source” on page 427. At that point, the master copy is retrieved from the specified URL and overwrites the local copy.

## Updating an Event Script from the Master Source

---

To update a single event script from its master source, issue the `set refresh` command at the `[edit event-options event-script file filename]` hierarchy level. The master source must already be configured as described in “Specifying a Master Source for an Event Script” on page 426.

```
[edit event-options event-script file filename]
user@host# set refresh
```

To update all enabled event scripts from their master sources, issue the `set refresh` command at the `[edit event-options event-script]` hierarchy level:

```
[edit event-options event-script]
user@host# set refresh
```

When you issue the `set refresh` command, the router immediately attempts to connect to the device that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local event scripts directory. If no master source for a script is defined, it is not updated and a warning is issued.

The update operation occurs as soon as you issue the `set refresh` command. The `refresh` statement is not added to the configuration. In other words, for this statement the `set` command behaves like an operational mode command, instead of adding a statement to the configuration.

If a platform has dual Routing Engines and you want the script to be updated 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.

## Updating an Event Script from an Alternate Location

---

In addition to updating an event script from the master source defined by the **source** statement at the `[edit event-options event-script file filename]` hierarchy level, you also can update a script from an alternate location. This is convenient when, for example, the master source cannot be accessed due to network or other problems. To update a single event script from the alternate source, issue the **set refresh-from** command at the `[edit event-options event-script file filename]` hierarchy level, specifying the location of the remote file:

```
[edit event-options event-script file filename]
user@host# set refresh-from url
```

To update all enabled event scripts from the alternate source, issue the **set refresh-from** command at the `[edit event-options event-script]` hierarchy level, specifying the location of the remote directory that houses the scripts:

```
[edit event-options event-script]
user@host# set refresh-from url
```

At both hierarchy levels:

**url**—URL of the remote event script or directory. Specify the source as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

When you issue the **set refresh-from** command, the router attempts to connect to the device that houses the master source for the script files and retrieve a copy of each file. The master copy overwrites the script stored in the local event scripts directory. If no master source for a script is defined, it is not updated and a warning is issued.

The update operation occurs as soon as you issue the **set refresh-from** command. The **refresh-from** statement is not added to the configuration. In other words, for this statement, the **set** command behaves like an operational mode command, instead of adding a statement to the configuration.

If a platform has dual Routing Engines and you want the script to be updated on both Routing Engines, you must issue the **set refresh-from** command on each Routing Engine separately. The **commit synchronize** command does not cause the **refresh-from** statement to update scripts on both Routing Engines.

## Converting an Event 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 “SLAX Overview” on page 27.

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

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/script1.slax
```

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. If you do not specify a filename for the destination file, it is named `SLAX-Conversion-Temp.xxxxx` where `xxxxx` is a randomly generated series of characters.

To convert a script from SLAX to XSLT, see “Converting an Event Script from SLAX to XSLT” on page 429.

## Converting an Event Script from SLAX to XSLT

---

To convert a SLAX script to XSLT, issue the `request system scripts convert slax-to-xslt` source *source-directory/source-filename* destination *destination-directory/<destination-filename>* operational mode command. 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/script1.xsl
```

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. If you do not specify a filename for the destination file, the file is named `SLAX-Conversion-Temp.xxxxx` where `xxxxx` is a randomly generated series of characters.

To convert a script from XSLT to SLAX, see “Converting an Event Script from XSLT to SLAX” on page 428.

## Tracing Event Script Processing

---

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

The default operation of event script tracing is to log important events in a file called `escript.log` located in the `/var/log` directory. When the file `escript.log` reaches 128 kilobytes (KB), it is renamed with a number 0 through 9 (in ascending order) appended to the end of the file and then compressed. The resulting files are `escript.log.0.gz`, then `escript.log.1.gz`, until there are 10 trace files. Then the oldest trace file (`escript.log.9.gz`) is overwritten. (For more information about how log files are created, see the *JUNOS System Log Messages Reference*.)

This section discusses the following topics:

- Minimum Configuration for Enabling Traceoptions for Event Scripts on page 430
- Configuring Tracing of Event Scripts on page 431

### **Minimum Configuration for Enabling Traceoptions for Event Scripts**

If no event script trace options are configured, the simplest way to view the trace output of an event script is to configure the `output` trace flag and issue the `show log escript.log | last` command. To do this, perform the following steps:

1. If you have not done so already, enable an event script by including the `file` statement at the `[edit event-options event-script]` hierarchy level:

```
[edit event-options event-script]
user@host# set file filename
```

2. Enable trace options by including the `traceoptions flag output` statement at the `[edit event-options event-script]` hierarchy level:

```
[edit event-options event-script]
user@host# set traceoptions flag output
```

3. Issue the `commit` command:

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

4. Display the resulting trace messages recorded in the `/var/log/escript.log` file. At the end of the log is the output generated by the event script you enabled in Step 1 after a configured event policy is triggered and invokes the script. To display the end of the log, issue the `show log escript.log | last` operational mode command:

```
[edit]
user@host# run show log escript.log | last
```

Table 21 on page 430 summarizes useful filtering commands that display selected portions of the `escript.log` file.

**Table 21: Event Script Tracing Operational Mode Commands**

| Task                                                              | Command                                               |
|-------------------------------------------------------------------|-------------------------------------------------------|
| Display logging data associated with all event script processing. | <code>show log escript.log</code>                     |
| Display processing for only the most recent operation.            | <code>show log escript.log   last</code>              |
| Display processing for script errors.                             | <code>show log escript.log   match error</code>       |
| Display processing for a particular script.                       | <code>show log escript.log   match script-name</code> |



### Example: Minimum Configuration for Enabling Traceoptions for Event Scripts

Display the trace output of the event script file `source-route.xml`:

```
[edit]
event-options {
 event-script {
 file source-route.xml;
 traceoptions flag output;
 }
}

[edit]
user@host# commit
[edit]
user@host# run show log escript.log | last
```

### Configuring Tracing of Event Scripts

You cannot change the directory (`/var/log`) to which trace files are written. However, you can customize other trace file settings by including the following statements at the `[edit event-options event-script traceoptions]` hierarchy level:

```
[edit event-options event-script traceoptions]
file <filename> <files number> <size size> <world-readable | no-world-readable>;
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
no-remote-trace;
```

These statements are described in the following sections:

- Configuring the Event Script Log Filename on page 431
- Configuring the Number and Size of Event Script Log Files on page 432
- Configuring Access to Event Script Log Files on page 432
- Configuring the Event Script Trace Operations on page 432

### Configuring the Event Script Log Filename

By default, the name of the file that records trace output is `escript.log`. You can specify a different name by including the `file` statement at the `[edit event-options event-script traceoptions]` hierarchy level:

```
[edit event-options event-script traceoptions]
file filename;
```

## Configuring the Number and Size of Event Script Log Files

By default, when the trace file reaches 128 KB in size, it is renamed and compressed to *filename.0.gz*, then *filename.1.gz*, and so on, until there are 10 trace files. Then the oldest trace file (*filename.9.gz*) is overwritten.

You can configure the limits on the number and size of trace files by including the following statements at the [edit event-options event-script traceoptions file <filename>] hierarchy level:

```
[edit event-options event-script traceoptions file <filename>]
files number size size;
```

For example, set the maximum file size to 640 KB and the maximum number of files to 20. When the file that receives the output of the tracing operation (*filename*) reaches 640 KB, it is renamed and compressed to *filename.0.gz*, and a new file called *filename* is created. When the new *filename* reaches 640 KB, *filename.0.gz* is renamed *filename.1.gz* and *filename* is renamed and compressed to *filename.0.gz*. This process repeats until there are 20 trace files. Then the oldest file (*filename.19.gz*) is overwritten.

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

If you set either a maximum file size or a maximum number of trace files, you also must specify the other parameter and a filename.

## Configuring Access to Event Script Log Files

By default, access to the event script log file is restricted to the owner. You can manually configure access by including the *world-readable* or *no-world-readable* statement at the [edit event-options event-script traceoptions file <filename>] hierarchy level.

```
[edit event-options event-script traceoptions file <filename>]
(world-readable | no-world-readable);
```

The *no-world-readable* statement restricts event script log access to the owner. The *world-readable* statement enables unrestricted access to the event script log file.

## Configuring the Event Script Trace Operations

By default, only important events are logged. You can configure the trace operations to be logged by including the following statements at the [edit event-options event-script traceoptions] hierarchy level:

```
[edit event-options event-script traceoptions]
flag all;
flag events;
flag input;
flag offline;
flag output;
flag rpc;
flag xslt;
```

Table 22 on page 433 describes the meaning of the event script tracing flags.

**Table 22: 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 29

# Event Script Examples

This chapter provides the following sample event scripts:

- Example: Limiting Event Script Output Based on a Specific Event Type on page 435

### Example: Limiting Event Script Output Based on a Specific Event Type

---

In situations where an event policy is triggered by multiple event types, you can limit the number of events that trigger the event script. For example, the following event policy triggers the `event-details.slax` event script whenever a `ui_login_event` or `ui_logout_event` occurs.

```
event-options {
 policy event-detail {
 events [ui_login_event ui_logout_event];
 then {
 event-script event-details.slax {
 output-filename systemlog;
 destination /tmp;
 }
 }
 }
}
```

The `event-details.slax` event script writes a log file only when the `ui_login_event` event occurs.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
ns ext = "http://xmlsoft.org/XSLT/namespace";

var $event-definition = {
 <event-options> {
 <policy> {
 <namex> "event-detail";
 <eventsx> "ui_login_event";
 <thenx> {
 <event-scriptx> {
 <namex> "event_detail.slax";
 <output-filenamex> "foo";
 <destinationx> {
```

```

 <namex> "foo";
 }
}
}
}
}
}

match / {
 <event-script-resultsx> {
 <event-triggered-this-policyx> {
 expr event-script-input/trigger-event/id;
 }
 <type-of-eventx> {
 expr event-script-input/trigger-event/type;
 }
 <process-namex> {
 expr event-script-input/trigger-event/attribute-list/attribute/name;
 }
 }
}

```

## Chapter 30

# Summary of Event Script Configuration Statements

This chapter describes each configuration statement for event scripts. The statements are organized alphabetically.

### event-script

---

**Syntax** event-script {  
    file *filename* {  
        refresh;  
        refresh-from *url*;  
        remote-execution {  
            remote-hostname {  
                passphrase *user-password*;  
                username *user-login*;  
            }  
        }  
        source *url*;  
    }  
    refresh;  
    refresh-from *url*;  
    traceoptions {  
        file <*filename*> <*files number*> <*size size*> <world-readable | no-world-readable>;  
        flag *flag*;  
        no-remote-trace;  
    }  
}

**Hierarchy Level** [edit event-options]

**Release Information** Statement introduced in JUNOS Release 7.6.

**Description** For JUNOS event scripts, configure scripting mechanisms.

The statements are explained separately.

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

**Related Topics** ■ Implementing Event Scripts on page 424

**file**

---

**Syntax**    file *filename* {  
               refresh;  
               refresh-from *url*;  
               remote-execution {  
                   *remote-hostname* {  
                       passphrase *user-password*;  
                       username *user-login*;  
                   }  
               }  
               source *url*;  
           }

**Hierarchy Level**    [edit event-options event-script]

**Release Information**    Statement introduced in JUNOS 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.

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

**Related Topics**    ■    Enabling an Event Script on page 425



## refresh

---

|                                 |                                                                                                                                                                                                                                                                                      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh;                                                                                                                                                                                                                                                                             |
| <b>Hierarchy Level</b>          | [edit event-options event-script],<br>[edit event-options event-script file <i>filename</i> ]                                                                                                                                                                                        |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 9.6.                                                                                                                                                                                                                                           |
| <b>Description</b>              | For JUNOS event scripts, overwrite the local copy of all enabled event scripts or a single enabled script located in the <code>/var/db/scripts/event</code> directory with the copy located at the source URL, specified in the <b>source</b> statement at the same hierarchy level. |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                                          |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh-from</li> <li>■ source</li> <li>■ Updating an Event Script from the Master Source on page 427</li> </ul>                                                                                                                            |

## refresh-from

---

|                                 |                                                                                                                                                                                                                                                                   |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | refresh-from <i>url</i> ;                                                                                                                                                                                                                                         |
| <b>Hierarchy Level</b>          | [edit event-options event-script],<br>[edit event-options event-script file <i>filename</i> ]                                                                                                                                                                     |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 9.6.                                                                                                                                                                                                                        |
| <b>Description</b>              | For JUNOS event scripts, overwrite the local copy of all enabled event scripts or a single enabled script located in the <code>/var/db/scripts/event</code> directory with the copy located at a URL other than the URL specified in the <b>source</b> statement. |
| <b>Options</b>                  | <i>url</i> —Source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.                                                                                                                          |
| <b>Required Privilege Level</b> | maintenance—To view this statement in the configuration.<br>maintenance-control—To add this statement to the configuration.                                                                                                                                       |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ refresh</li> <li>■ source</li> <li>■ Updating an Event Script from an Alternate Location on page 428</li> </ul>                                                                                                          |

## remote-execution

---

**Syntax**    `remote-execution {  
                   remote-hostname {  
                     passphrase user-password;  
                     username user-login;  
                   }  
                 }`

**Hierarchy Level**    [edit event-options event-script file *filename*]

**Release Information**    Statement introduced in JUNOS Release 9.6.

**Description**    For JUNOS event scripts, enable event scripts to invoke RPCs on a local or remote host.

**Options**    *remote-hostname*—Name of the remote host with which the event script will communicate.

*passphrase user-password*—User's password for the remote host.

*username username*—User's login name for the remote host.

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

**Related Topics**    ■    Using RPCs and Operational Mode Commands in Event Scripts on page 417

**source**

---

|                                 |                                                                                                                                                                                                                                                                                                          |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                   | <code>source url;</code>                                                                                                                                                                                                                                                                                 |
| <b>Hierarchy Level</b>          | [edit event-options event-script file <i>filename</i> ]                                                                                                                                                                                                                                                  |
| <b>Release Information</b>      | Statement introduced in JUNOS Release 9.6.                                                                                                                                                                                                                                                               |
| <b>Description</b>              | For JUNOS event scripts, specify the location of the source file for an enabled script located in the <code>/var/db/scripts/event</code> directory. When you include the <b>refresh</b> statement at the same hierarchy level, the local copy is overwritten by the version stored at the specified URL. |
| <b>Options</b>                  | <i>url</i> —Master source file for an event script specified as an HTTP URL, FTP URL, or scp-style remote file specification.                                                                                                                                                                            |
| <b>Required Privilege Level</b> | <p><code>maintenance</code>—To view this statement in the configuration.</p> <p><code>maintenance-control</code>—To add this statement to the configuration.</p>                                                                                                                                         |
| <b>Related Topics</b>           | <ul style="list-style-type: none"> <li>■ <code>refresh</code></li> <li>■ <code>refresh-from</code></li> <li>■ Specifying a Master Source for an Event Script on page 426</li> </ul>                                                                                                                      |

## traceoptions

---

**Syntax**    `traceoptions {  
               file <filename> <files number> <size size> <world-readable | no-world-readable>;  
               flag flag;  
               no-remote-trace;  
           }`

**Hierarchy Level**    [edit event-options event-script]

**Release Information**    Statement introduced in JUNOS 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 `escript.log`. If you include the `file` statement, you must specify a filename. To retain the default, you can specify `escript.log` as the filename.

*files number*—(Optional) Maximum number of trace files. When a trace file named *trace-file* reaches its maximum size, it is renamed and compressed to *trace-file.0.gz*, then *trace-file.1.gz*, 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

`no-world-readable`—Restrict file access to owner. This is the default.

*size size*—(Optional) Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named *trace-file* reaches this size, it is

renamed and compressed to *trace-file.0.gz*. When *trace-file* again reaches its maximum size, *trace-file.0.gz* is renamed *trace-file.1.gz* and *trace-file* is renamed and compressed to *trace-file.0.gz*. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option and a filename.

**Syntax:** *xk* to specify KB, *xm* to specify MB, or *xg* to specify GB

**Range:** 10 KB through 1 GB

**Default:** 128 KB

**world-readable**—Enable unrestricted file access.

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

**Related Topics** ■ Tracing Event Script Processing on page 429



## **Part 6**

# **Index**

- Index on page 447
- Index of Statements and Commands on page 457





# Index

## Symbols

|                                              |       |
|----------------------------------------------|-------|
| #, comments in configuration statements..... | xxxii |
| \$                                           |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| ( ), in syntax descriptions.....             | xxxii |
| *                                            |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| +                                            |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| .                                            |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| < >, in syntax descriptions.....             | xxxii |
| ?                                            |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| [ ], in configuration statements.....        | xxxii |
| ^                                            |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| { }, in configuration statements.....        | xxxii |
| (pipe)                                       |       |
| regular expression operator                  |       |
| event policy.....                            | 351   |
| (pipe), in syntax descriptions.....          | xxxii |

## A

|                            |     |
|----------------------------|-----|
| adding                     |     |
| default encapsulation type |     |
| commit script example..... | 209 |
| final firewall term        |     |
| commit script example..... | 216 |
| interface to RIP group     |     |
| commit script example..... | 206 |
| all (tracing flag)         |     |
| commit scripts.....        | 180 |
| event policy.....          | 369 |
| event scripts.....         | 432 |
| op scripts.....            | 308 |

|                                      |     |
|--------------------------------------|-----|
| allow-transients statement.....      | 277 |
| usage guidelines.....                | 141 |
| apply-macro statement.....           | 278 |
| usage guidelines.....                | 154 |
| apply-templates SLAX statement.....  | 91  |
| applying templates                   |     |
| SLAX.....                            | 34  |
| XSLT.....                            | 19  |
| archive-sites statement.....         | 383 |
| usage guidelines.....                | 353 |
| archiving files in event policy..... | 353 |
| arguments statement                  |     |
| event policy.....                    | 384 |
| usage guidelines.....                | 360 |
| op scripts.....                      | 333 |
| usage guidelines.....                | 295 |
| assigning CoS classifier             |     |
| commit script example.....           | 252 |
| attributes                           |     |
| SLAX.....                            | 33  |
| XML in customized messages.....      | 127 |
| XSLT.....                            | 64  |
| attributes-match statement.....      | 384 |
| usage guidelines.....                | 347 |

## B

|                                          |       |
|------------------------------------------|-------|
| boilerplate                              |       |
| commit scripts.....                      | 115   |
| event scripts.....                       | 415   |
| op scripts.....                          | 291   |
| braces, in configuration statements..... | xxxii |
| brackets                                 |       |
| angle, in syntax descriptions.....       | xxxii |
| square, in configuration statements..... | xxxii |

## C

|                              |          |
|------------------------------|----------|
| call SLAX statement.....     | 92       |
| < change > XSLT element..... | 270      |
| usage guidelines.....        | 141, 154 |
| command statement.....       | 334      |
| usage guidelines.....        | 300      |
| commands statement.....      | 385      |
| usage guidelines.....        | 357      |

|                                                  |       |
|--------------------------------------------------|-------|
| comments                                         |       |
| SLAX and XSLT.....                               | 36    |
| comments, in configuration statements.....       | xxxii |
| commit script examples                           |       |
| adding default encapsulation type.....           | 209   |
| adding final firewall term.....                  | 216   |
| adding interface to RIP group.....               | 206   |
| assigning CoS classifier.....                    | 252   |
| configuring dual Routing Engines.....            | 234   |
| controlling minimum MTU.....                     | 189   |
| controlling routing table imports.....           | 238   |
| decreasing manual configuration.....             | 203   |
| explained line-by-line.....                      | 119   |
| generating error messages.....                   | 132   |
| generating persistent configuration changes..... | 148   |
| generating system log messages.....              | 135   |
| generating transient configuration changes.....  | 150   |
| generating warning messages.....                 | 129   |
| limiting number of ATM VCs.....                  | 200   |
| limiting number of interfaces.....               | 191   |
| prohibiting configuration statements.....        | 183   |
| reordering routing policies.....                 | 247   |
| requiring configuration statements.....          | 183   |
| requiring internal clocking.....                 | 187   |
| commit scripts                                   |       |
| attributes for customized messages.....          | 127   |
| boilerplate.....                                 | 115   |
| commands for monitoring.....                     | 176   |
| configuration statement summaries.....           | 277   |
| deactivating.....                                | 168   |
| deleting.....                                    | 168   |
| design considerations.....                       | 117   |
| enabling.....                                    | 168   |
| error messages, generating.....                  | 123   |
| examples <i>See</i> commit script examples       |       |
| extension functions                              |       |
| summaries.....                                   | 76    |
| usage guidelines.....                            | 39    |
| flow of operation illustrated.....               | 110   |
| input and output illustrated.....                | 56    |
| macros.....                                      | 153   |
| flow of operation illustrated.....               | 153   |
| making optional.....                             | 168   |
| master source                                    |       |
| configuring.....                                 | 173   |
| updating from.....                               | 173   |
| multiple.....                                    | 114   |
| named templates.....                             | 87    |
| output, displaying.....                          | 176   |
| overview.....                                    | 107   |
| persistent configuration changes.....            | 137   |
| remote sources                                   |       |
| overview.....                                    | 171   |
| updating from.....                               | 171   |
| specifying storage location.....                 | 171   |
| super-user login class, necessity of.....        | 168   |
| system log messages, generating.....             | 123   |
| trace log files.....                             | 177   |
| tracing flags.....                               | 180   |
| transient configuration changes.....             | 137   |
| troubleshooting.....                             | 181   |
| updating                                         |       |
| from alternate location.....                     | 174   |
| from master source.....                          | 173   |
| using multiple.....                              | 114   |
| warning messages, generating.....                | 123   |
| commit statement.....                            | 279   |
| usage guidelines.....                            | 168   |
| concat() XSLT function.....                      | 60    |
| concatenating XPath arguments in SLAX.....       | 32    |
| configuration                                    |       |
| dual Routing Engines (commit script              |       |
| example).....                                    | 234   |
| generating persistent changes to.....            | 141   |
| example.....                                     | 148   |
| generating transient changes to.....             | 141   |
| example.....                                     | 150   |
| configuration (event policy tracing flag).....   | 369   |
| configuration mode commands                      |       |
| commit script.....                               | 176   |
| contains() XSLT function.....                    | 60    |
| context node.....                                | 25    |
| controlling                                      |       |
| minimum MTU                                      |       |
| commit script example.....                       | 189   |
| routing table imports                            |       |
| commit script example.....                       | 238   |
| conventions                                      |       |
| text and syntax.....                             | xxxix |
| converting                                       |       |
| SLAX scripts to XSLT                             |       |
| commit scripts.....                              | 175   |
| event scripts.....                               | 429   |
| op scripts.....                                  | 305   |
| overview.....                                    | 29    |
| XSLT scripts to SLAX                             |       |
| commit scripts.....                              | 175   |
| event scripts.....                               | 428   |
| op scripts.....                                  | 304   |
| overview.....                                    | 29    |
| correlating events in event policy.....          | 347   |
| example                                          |       |
| based on attributes.....                         | 379   |
| representing.....                                | 374   |
| within time interval.....                        | 371   |
| count() XSLT function.....                       | 60    |
| curly braces, in configuration statements.....   | xxxii |
| customer support.....                            | xxxii |
| contacting JTAC.....                             | xxxii |

|                        |     |
|------------------------|-----|
| customizing            |     |
| show command output    |     |
| op script example..... | 316 |
| show commands          |     |
| op script example..... | 313 |

## D

|                                             |       |
|---------------------------------------------|-------|
| database (event policy tracing flag).....   | 369   |
| deactivating                                |       |
| scripts in the configuration.....           | 168   |
| decreasing manual configuration             |       |
| commit script example.....                  | 203   |
| delaying file transfer by event policy..... | 353   |
| deleting                                    |       |
| scripts from the configuration.....         | 168   |
| description statement                       |       |
| op script arguments.....                    | 334   |
| usage guidelines.....                       | 295   |
| op scripts.....                             | 334   |
| usage guidelines.....                       | 298   |
| destination statement                       |       |
| event policy.....                           | 409   |
| usage guidelines for command                |       |
| execution.....                              | 357   |
| usage guidelines for event script           |       |
| execution.....                              | 360   |
| usage guidelines for file upload.....       | 354   |
| destinations statement.....                 | 387   |
| usage guidelines.....                       | 353   |
| direct-access statement.....                | 279   |
| usage guidelines.....                       | 174   |
| documentation                               |       |
| comments on.....                            | xxxii |
| dot node.....                               | 25    |
| DTD                                         |       |
| defined.....                                | 11    |

## E

|                                             |     |
|---------------------------------------------|-----|
| elements                                    |     |
| SLAX.....                                   | 32  |
| XSLT <i>See</i> XSLT elements               |     |
| else if SLAX statement.....                 | 94  |
| usage guidelines.....                       | 29  |
| else SLAX statement.....                    | 94  |
| usage guidelines.....                       | 29  |
| equals statement.....                       | 387 |
| usage guidelines.....                       | 347 |
| error messages, generating custom.....      | 123 |
| example.....                                | 132 |
| event policy                                |     |
| changing privilege level for execution..... | 366 |
| configuration statement summaries.....      | 383 |
| configuring destinations.....               | 353 |
| configuring file transfer delays.....       | 353 |

|                                                |          |
|------------------------------------------------|----------|
| correlating events.....                        | 347      |
| delaying file upload.....                      | 355      |
| event details                                  |          |
| received events.....                           | 420      |
| remote execution details.....                  | 420      |
| triggered events.....                          | 420      |
| example <i>See</i> event policy examples       |          |
| executing commands.....                        | 357      |
| executing op scripts.....                      | 360      |
| flow of operation illustrated.....             | 341      |
| generating events.....                         | 351      |
| ignoring events.....                           | 365      |
| overview.....                                  | 341      |
| raising SNMP traps.....                        | 366      |
| regular expression filtering.....              | 350      |
| retrying file upload.....                      | 356      |
| tracing flags.....                             | 369      |
| tracing operations.....                        | 367      |
| triggering based on event count.....           | 350      |
| triggering by nonstandard system log           |          |
| messages.....                                  | 352      |
| uploading event files.....                     | 354      |
| event policy examples                          |          |
| changing privilege level for execution.....    | 374      |
| configuring transfer delay.....                | 372      |
| correlating events                             |          |
| based on attributes.....                       | 379      |
| representing.....                              | 374      |
| within time interval.....                      | 371      |
| generating internal events.....                | 380, 381 |
| ignoring events.....                           | 378      |
| raising SNMP traps.....                        | 381      |
| regular expression filtering.....              | 380      |
| retrying file upload.....                      | 375      |
| triggering based on event count.....           | 377      |
| triggering with nonstandard system log         |          |
| message.....                                   | 381      |
| event script examples                          |          |
| limiting policy trigger to specific event..... | 435      |
| event scripts                                  |          |
| boilerplate.....                               | 415      |
| configuration statement summaries.....         | 437      |
| configuring.....                               | 424      |
| enabling.....                                  | 425      |
| examples <i>See</i> event script examples      |          |
| executing.....                                 | 426      |
| extension functions                            |          |
| summaries.....                                 | 76       |
| usage guidelines.....                          | 39       |
| master source                                  |          |
| configuring.....                               | 426      |
| updating from.....                             | 427      |
| named templates.....                           | 87       |
| overview.....                                  | 413      |
| replacing.....                                 | 424      |
| specifying storage location.....               | 426      |

|                                                         |     |
|---------------------------------------------------------|-----|
| super-user login class, necessity of.....               | 424 |
| trace log files.....                                    | 429 |
| tracing flags.....                                      | 432 |
| updating.....                                           |     |
| from alternate location.....                            | 428 |
| from master source.....                                 | 427 |
| using.....                                              | 413 |
| writing.....                                            | 415 |
| event-options statement.....                            | 388 |
| usage guidelines.....                                   | 345 |
| event-script statement.....                             |     |
| defining script.....                                    | 437 |
| usage guidelines.....                                   | 417 |
| invoking script in event policy.....                    | 390 |
| usage guidelines.....                                   | 360 |
| events (tracing flag).....                              |     |
| commit scripts.....                                     | 180 |
| event policy.....                                       | 369 |
| event scripts.....                                      | 432 |
| op scripts.....                                         | 308 |
| events statement.....                                   | 391 |
| usage guidelines.....                                   | 347 |
| examples.....                                           |     |
| commit scripts <i>See</i> commit script examples        |     |
| event policy <i>See</i> event policy examples           |     |
| event scripts <i>See</i> event script examples          |     |
| op scripts <i>See</i> op script examples                |     |
| execute-commands statement.....                         | 393 |
| usage guidelines.....                                   | 357 |
| executing operational-mode commands.....                | 357 |
| expr statement in SLAX.....                             | 32  |
| expressions in SLAX.....                                | 32  |
| extension functions <i>See</i> XSLT extension functions |     |

## F

|                                            |      |
|--------------------------------------------|------|
| file statement.....                        |      |
| commit scripts.....                        | 280  |
| usage guidelines.....                      | 168  |
| event scripts.....                         | 438  |
| usage guidelines.....                      | 425  |
| op scripts.....                            | 335  |
| usage guidelines.....                      | 300  |
| filename statement.....                    |      |
| event policy.....                          | 409  |
| usage guidelines.....                      | 354  |
| finding LSPs to multiple destinations..... |      |
| op script example.....                     | 326  |
| font conventions.....                      | xxxi |
| for-each SLAX statement.....               | 95   |
| usage guidelines.....                      | 29   |
| functions, XSLT <i>See</i> XSLT functions  |      |

## G

|                                 |          |
|---------------------------------|----------|
| generate-event statement.....   | 394      |
| usage guidelines.....           | 351      |
| generating internal events..... | 351      |
| example.....                    | 380, 381 |

## I

|                                      |     |
|--------------------------------------|-----|
| icons defined, notice.....           | xxx |
| if SLAX statement.....               | 96  |
| usage guidelines.....                | 29  |
| ignore statement.....                | 394 |
| usage guidelines.....                | 365 |
| ignoring events in event policy..... | 365 |
| example.....                         | 378 |
| input (tracing flag).....            |     |
| commit scripts.....                  | 180 |
| event scripts.....                   | 432 |
| op scripts.....                      | 308 |

## J

|                                       |    |
|---------------------------------------|----|
| jcs:break-lines() XSLT function.....  | 76 |
| usage guidelines.....                 | 40 |
| jcs:close() XSLT function.....        | 76 |
| usage guidelines.....                 | 40 |
| jcs:dampen() XSLT function.....       | 77 |
| usage guidelines.....                 | 41 |
| <jcs:edit-path> XSLT template.....    | 87 |
| usage guidelines.....                 | 51 |
| <jcs:emit-change> XSLT template.....  | 88 |
| usage guidelines.....                 | 52 |
| <jcs:emit-comment> XSLT template..... | 89 |
| usage guidelines.....                 | 54 |
| jcs:empty() XSLT function.....        | 77 |
| usage guidelines.....                 | 41 |
| jcs:execute() XSLT function.....      | 78 |
| usage guidelines.....                 | 42 |
| jcs:first-of() XSLT function.....     | 78 |
| usage guidelines.....                 | 42 |
| jcs:get-input() XSLT function.....    | 78 |
| usage guidelines.....                 | 43 |
| jcs:get-secret() XSLT function.....   | 79 |
| usage guidelines.....                 | 43 |
| jcs:hostname() XSLT function.....     | 79 |
| usage guidelines.....                 | 43 |
| jcs:invoke() XSLT function.....       | 80 |
| usage guidelines.....                 | 44 |
| jcs:open() XSLT function.....         | 80 |
| usage guidelines.....                 | 44 |
| jcs:output() XSLT function.....       | 81 |
| usage guidelines.....                 | 45 |
| jcs:parse-ip() XSLT function.....     | 82 |
| usage guidelines.....                 | 45 |
| jcs:printf() XSLT function.....       | 83 |
| usage guidelines.....                 | 46 |

|                                                               |     |
|---------------------------------------------------------------|-----|
| jcs:progress() XSLT function.....                             | 83  |
| usage guidelines.....                                         | 46  |
| jcs:regex() XSLT function.....                                | 84  |
| usage guidelines.....                                         | 46  |
| jcs:sleep() XSLT function.....                                | 84  |
| usage guidelines.....                                         | 47  |
| jcs:split() XSLT function.....                                | 85  |
| usage guidelines.....                                         | 47  |
| <jcs:statement> XSLT template.....                            | 89  |
| usage guidelines.....                                         | 55  |
| jcs:sysctl() XSLT function.....                               | 85  |
| usage guidelines.....                                         | 48  |
| jcs:syslog() XSLT function.....                               | 86  |
| usage guidelines.....                                         | 49  |
| jcs:trace() XSLT function.....                                | 86  |
| usage guidelines.....                                         | 50  |
| JUNOS extension functions <i>See</i> XSLT extension functions |     |
| JUNOS named templates.....                                    | 87  |
| JUNOS XML API                                                 |     |
| overview.....                                                 | 9   |
| JUNOS XML RPCs                                                |     |
| sample use in op script.....                                  | 316 |
| JUNOS XML tags                                                |     |
| notational conventions.....                                   | 10  |
| junos.xml file                                                |     |
| importing.....                                                | 51  |
| templates in                                                  |     |
| summaries.....                                                | 87  |
| usage guidelines.....                                         | 51  |
| JUNOScript API                                                |     |
| advantages of.....                                            | 11  |
| overview.....                                                 | 9   |
| JUNOScript server.....                                        | 9   |
| JUNOScript session                                            |     |
| brief overview.....                                           | 12  |
| JUNOScript tags                                               |     |
| notational conventions.....                                   | 10  |

## K

|                               |     |
|-------------------------------|-----|
| KERNEL system log messages    |     |
| trigger for event policy..... | 352 |

## L

|                               |     |
|-------------------------------|-----|
| last() XSLT function.....     | 61  |
| LCC system log messages       |     |
| trigger for event policy..... | 352 |
| limiting number of ATM VCs    |     |
| commit script example.....    | 200 |
| limiting number of interfaces |     |
| commit script example.....    | 191 |

|                                   |     |
|-----------------------------------|-----|
| load-scripts-from-flash statement |     |
| usage guidelines                  |     |
| commit scripts.....               | 171 |
| event scripts.....                | 426 |
| op scripts.....                   | 302 |
| loading a base configuration      |     |
| commit script example.....        | 255 |

## M

|                                           |       |
|-------------------------------------------|-------|
| macros in commit scripts                  |       |
| advantages of.....                        | 159   |
| example                                   |       |
| creating MPLS group.....                  | 161   |
| loading a base configuration.....         | 255   |
| simplifying IGP configuration.....        | 220   |
| simplifying interface configuration.....  | 224   |
| simplifying IP address configuration..... | 241   |
| simplifying LDP configuration.....        | 212   |
| simplifying MPLS LSP configuration.....   | 230   |
| overview.....                             | 153   |
| manuals                                   |       |
| comments on.....                          | xxxii |
| match SLAX statement.....                 | 97    |
| usage guidelines.....                     | 30    |
| matches statement.....                    | 395   |
| usage guidelines.....                     | 350   |
| mode SLAX statement.....                  | 98    |
| multiple commit scripts.....              | 114   |

## N

|                           |     |
|---------------------------|-----|
| name() XSLT function..... | 61  |
| named templates           |     |
| SLAX.....                 | 35  |
| XSLT.....                 | 19  |
| not statement.....        | 396 |
| usage guidelines.....     | 347 |
| not() XSLT function.....  | 61  |
| notice icons defined..... | xxx |
| ns SLAX statement         |     |
| usage guidelines.....     | 31  |

## O

|                                            |     |
|--------------------------------------------|-----|
| offline (tracing flag)                     |     |
| commit scripts.....                        | 180 |
| event scripts.....                         | 432 |
| op scripts.....                            | 308 |
| op script examples                         |     |
| customizing show command output.....       | 316 |
| finding LSPs to multiple destinations..... | 326 |
| restarting an FPC.....                     | 311 |
| simplifying show command.....              | 313 |

|                                            |          |  |
|--------------------------------------------|----------|--|
| op scripts                                 |          |  |
| alias, defining.....                       | 300      |  |
| arguments, declaring.....                  | 295      |  |
| boilerplate.....                           | 291      |  |
| configuration statement summaries.....     | 333      |  |
| configuring.....                           | 299      |  |
| enabling.....                              | 300      |  |
| examples <i>See</i> op script examples     |          |  |
| executing.....                             | 301      |  |
| extension functions                        |          |  |
| summaries.....                             | 76       |  |
| usage guidelines.....                      | 39       |  |
| flow of operation illustrated.....         | 290      |  |
| help text, configuring.....                | 298      |  |
| master source                              |          |  |
| specifying.....                            | 302      |  |
| updating from.....                         | 303      |  |
| named templates.....                       | 87       |  |
| overview.....                              | 289      |  |
| specifying storage location.....           | 302      |  |
| super-user login class, necessity of.....  | 300      |  |
| trace log files.....                       | 305      |  |
| tracing flags.....                         | 308      |  |
| updating                                   |          |  |
| from alternate location.....               | 304      |  |
| from master source.....                    | 303      |  |
| using.....                                 | 289      |  |
| writing.....                               | 291      |  |
| op statement.....                          | 336      |  |
| usage guidelines.....                      | 300      |  |
| operational mode commands                  |          |  |
| displaying output from commit scripts..... | 176      |  |
| event scripts                              |          |  |
| displaying output fields as XML.....       | 417      |  |
| invoking.....                              | 419      |  |
| without XML equivalent.....                | 419      |  |
| op scripts                                 |          |  |
| displaying output fields as XML.....       | 293      |  |
| invoking.....                              | 294      |  |
| without XML equivalent.....                | 294      |  |
| operators, regular expression              |          |  |
| event policy.....                          | 350      |  |
| example.....                               | 380      |  |
| optional statement.....                    | 280      |  |
| usage guidelines.....                      | 168      |  |
| output (tracing flag)                      |          |  |
| commit scripts.....                        | 180      |  |
| event scripts.....                         | 432      |  |
| op scripts.....                            | 308      |  |
| output-filename statement.....             | 396      |  |
| usage guidelines.....                      | 357, 360 |  |
| output-format statement.....               | 397      |  |
| usage guidelines.....                      | 357, 360 |  |
| overview                                   |          |  |
| commit scripts.....                        | 107      |  |
| event policy.....                          | 341      |  |
| event scripts.....                         | 413      |  |
| op scripts.....                            | 289      |  |
| SLAX.....                                  | 27       |  |
| XML.....                                   | 10       |  |
| XSLT.....                                  | 15       |  |
| <b>P</b>                                   |          |  |
| param SLAX statement.....                  | 99       |  |
| parameters                                 |          |  |
| SLAX                                       |          |  |
| declaring.....                             | 33       |  |
| passing to templates.....                  | 35       |  |
| XSLT.....                                  | 20       |  |
| parentheses, in syntax descriptions.....   | xxxii    |  |
| persistent configuration changes           |          |  |
| compared to transient changes.....         | 137      |  |
| example.....                               | 148      |  |
| generating.....                            | 141      |  |
| overview.....                              | 137      |  |
| removing.....                              | 145      |  |
| tags and attributes for.....               | 147      |  |
| PFE system log messages                    |          |  |
| trigger for event policy.....              | 352      |  |
| PIC system log messages                    |          |  |
| trigger for event policy.....              | 352      |  |
| policy (event policy tracing flag).....    | 369      |  |
| policy statement.....                      | 398      |  |
| usage guidelines.....                      | 345      |  |
| position() XSLT function.....              | 62       |  |
| postinheritance, defined.....              | 110      |  |
| priority SLAX statement.....               | 100      |  |
| programming instructions, XSLT             |          |  |
| < xsl:choose > .....                       | 23       |  |
| < xsl:for-each > .....                     | 23       |  |
| < xsl:if > .....                           | 23       |  |
| prohibiting configuration statements       |          |  |
| commit script example.....                 | 183      |  |
| <b>R</b>                                   |          |  |
| raise-trap statement.....                  | 400      |  |
| usage guidelines.....                      | 366      |  |
| recursion, XSLT.....                       | 25       |  |
| refresh operation                          |          |  |
| commit scripts.....                        | 173      |  |
| event scripts.....                         | 427      |  |
| op scripts.....                            | 303      |  |
| refresh statement                          |          |  |
| commit scripts.....                        | 281      |  |
| usage guidelines.....                      | 173      |  |
| event scripts.....                         | 439      |  |
| usage guidelines.....                      | 427      |  |
| op scripts.....                            | 337      |  |
| usage guidelines.....                      | 303      |  |

|                                                     |     |
|-----------------------------------------------------|-----|
| refresh-from statement                              |     |
| commit scripts.....                                 | 281 |
| usage guidelines.....                               | 174 |
| event scripts.....                                  | 439 |
| usage guidelines.....                               | 428 |
| op scripts.....                                     | 337 |
| usage guidelines.....                               | 304 |
| regular expression operators                        |     |
| event policy.....                                   | 350 |
| example.....                                        | 380 |
| remote source for commit scripts                    |     |
| overview.....                                       | 171 |
| updating from.....                                  | 171 |
| remote-execution statement.....                     | 440 |
| usage guidelines.....                               | 418 |
| reordering routing policies                         |     |
| commit script example.....                          | 247 |
| request system scripts event-scripts reload command |     |
| usage guidelines.....                               | 424 |
| requiring configuration statements                  |     |
| commit script example.....                          | 183 |
| requiring internal clocking                         |     |
| commit script example.....                          | 187 |
| retry-count statement.....                          | 400 |
| usage guidelines.....                               | 356 |
| retry-interval statement.....                       | 400 |
| usage guidelines.....                               | 356 |
| rpc (tracing flag)                                  |     |
| commit scripts.....                                 | 180 |
| event scripts.....                                  | 432 |
| op scripts.....                                     | 308 |
| RPCs                                                |     |
| event scripts                                       |     |
| displaying output fields.....                       | 417 |
| invoking.....                                       | 418 |
| op scripts                                          |     |
| displaying output fields.....                       | 293 |
| example.....                                        | 316 |
| invoking.....                                       | 294 |
| <b>S</b>                                            |     |
| SCC system log messages                             |     |
| trigger for event policy.....                       | 352 |
| scripts statement.....                              | 282 |
| usage guidelines.....                               | 168 |
| server <i>See</i> JUNOScript server                 |     |
| server (event policy tracing flag).....             | 369 |
| session <i>See</i> JUNOScript session               |     |
| simplifying                                         |     |
| IGP configuration                                   |     |
| commit script example.....                          | 220 |
| interface configuration                             |     |
| commit script example.....                          | 224 |
| IP address configuration                            |     |
| commit script example.....                          | 241 |

|                                       |     |
|---------------------------------------|-----|
| LDP configuration                     |     |
| commit script example.....            | 212 |
| MPLS LSP configuration                |     |
| commit script example.....            | 230 |
| SLAX                                  |     |
| applying templates.....               | 34  |
| attributes.....                       | 33  |
| benefits of.....                      | 28  |
| comments.....                         | 36  |
| converting to XSLT                    |     |
| commit scripts.....                   | 175 |
| event scripts.....                    | 429 |
| op scripts.....                       | 305 |
| overview.....                         | 29  |
| elements.....                         | 32  |
| expr statement.....                   | 32  |
| expressions.....                      | 32  |
| flow of operation illustrated.....    | 28  |
| named templates.....                  | 35  |
| overview.....                         | 27  |
| parameters.....                       | 33  |
| purpose.....                          | 28  |
| statements <i>See</i> SLAX statements |     |
| template parameters.....              | 35  |
| using the XSL namespace.....          | 37  |
| using XSLT elements.....              | 37  |
| variables.....                        | 33  |
| SLAX statements                       |     |
| apply-templates.....                  | 91  |
| call.....                             | 92  |
| else.....                             | 94  |
| usage guidelines.....                 | 29  |
| else if.....                          | 94  |
| usage guidelines.....                 | 29  |
| for-each.....                         | 95  |
| usage guidelines.....                 | 29  |
| if.....                               | 96  |
| usage guidelines.....                 | 29  |
| match.....                            | 97  |
| usage guidelines.....                 | 30  |
| mode.....                             | 98  |
| ns                                    |     |
| usage guidelines.....                 | 31  |
| param.....                            | 99  |
| priority.....                         | 100 |
| template.....                         | 101 |
| var.....                              | 102 |
| version.....                          | 103 |
| usage guidelines.....                 | 31  |
| with.....                             | 104 |
| SNMP traps                            |     |
| raising in event policy.....          | 366 |
| example.....                          | 381 |

|                                                 |      |
|-------------------------------------------------|------|
| source statement                                |      |
| commit scripts.....                             | 283  |
| usage guidelines.....                           | 173  |
| event scripts.....                              | 441  |
| usage guidelines.....                           | 426  |
| op scripts.....                                 | 338  |
| usage guidelines.....                           | 302  |
| starts-with statement.....                      | 401  |
| usage guidelines.....                           | 347  |
| starts-with() XSLT function.....                | 62   |
| statements in SLAX <i>See</i> SLAX statements   |      |
| string-length() XSLT function.....              | 62   |
| substring-after() XSLT function.....            | 63   |
| substring-before() XSLT function.....           | 63   |
| super-user login class                          |      |
| necessity of for commit scripts.....            | 168  |
| necessity of for event scripts.....             | 424  |
| necessity of for op scripts.....                | 300  |
| support, technical <i>See</i> technical support |      |
| syntax conventions.....                         | xxxi |
| < syslog > JUNOS XML tag.....                   | 271  |
| usage guidelines.....                           | 124  |
| syslogd (event policy tracing flag).....        | 369  |
| system log messages                             |      |
| generated by commit script.....                 | 123  |
| example.....                                    | 135  |
| trigger for event policy.....                   | 352  |
| SYSTEM system log messages                      |      |
| trigger for event policy.....                   | 352  |

## T

|                                                 |      |
|-------------------------------------------------|------|
| tags <i>See</i> JUNOS XML tags, JUNOScript tags |      |
| tags for customized messages.....               | 127  |
| technical support                               |      |
| contacting JTAC.....                            | xxxi |
| template SLAX statement.....                    | 101  |
| templates <i>See</i> XSLT templates             |      |
| applying in SLAX.....                           | 34   |
| named                                           |      |
| SLAX.....                                       | 35   |
| XSLT.....                                       | 19   |
| parameters in SLAX.....                         | 35   |
| unnamed XSLT.....                               | 19   |
| XSLT.....                                       | 18   |
| then statement.....                             | 402  |
| usage guidelines.....                           | 345  |
| time-interval statement.....                    | 404  |
| usage guidelines.....                           | 351  |
| time-of-day statement.....                      | 404  |
| usage guidelines.....                           | 351  |
| timer-events (event policy tracing flag).....   | 369  |

|                                        |          |
|----------------------------------------|----------|
| traceoptions statement                 |          |
| commit scripts.....                    | 284      |
| usage guidelines.....                  | 177      |
| event policy.....                      | 405      |
| usage guidelines.....                  | 367      |
| event scripts.....                     | 442      |
| usage guidelines.....                  | 429      |
| op scripts.....                        | 284      |
| usage guidelines.....                  | 305      |
| tracing flags.....                     | 180      |
| commit scripts.....                    | 180      |
| event policy.....                      | 369      |
| event scripts.....                     | 432      |
| op scripts.....                        | 308      |
| <i>See also</i> entries for flag names |          |
| tracing operations                     |          |
| commit scripts.....                    | 177      |
| event policy.....                      | 367      |
| event scripts.....                     | 429      |
| op scripts.....                        | 305      |
| transfer delay in event policy         |          |
| example.....                           | 372      |
| transfer-delay statement.....          | 407      |
| usage guidelines                       |          |
| event policy.....                      | 355      |
| specific destination.....              | 353      |
| transient configuration changes        |          |
| compared to persistent changes.....    | 137      |
| example.....                           | 150      |
| generating.....                        | 141      |
| overview.....                          | 137      |
| removing.....                          | 145      |
| tags and attributes for.....           | 147      |
| < transient-change > XSLT element..... | 272      |
| usage guidelines.....                  | 141, 154 |
| traps, SNMP                            |          |
| raising in event policy.....           | 366      |
| example.....                           | 381      |
| trigger statement.....                 | 408      |
| usage guidelines.....                  | 350      |
| troubleshooting commit scripts.....    | 181      |

## U

|                              |     |
|------------------------------|-----|
| unnamed XSLT templates.....  | 19  |
| updating                     |     |
| commit scripts               |     |
| from alternate location..... | 174 |
| from master source.....      | 173 |
| event scripts                |     |
| from alternate location..... | 428 |
| from master source.....      | 427 |
| op scripts                   |     |
| from alternate location..... | 304 |
| from master source.....      | 303 |



|                            |     |
|----------------------------|-----|
| upload statement.....      | 409 |
| usage guidelines.....      | 354 |
| uploading event files..... | 354 |
| user-name statement.....   | 410 |
| usage guidelines.....      | 366 |

## V

|                             |     |
|-----------------------------|-----|
| var SLAX statement.....     | 102 |
| variables                   |     |
| SLAX.....                   | 33  |
| XSLT.....                   | 21  |
| version SLAX statement..... | 103 |
| usage guidelines.....       | 31  |

## W

|                                          |     |
|------------------------------------------|-----|
| warning messages, generating custom..... | 123 |
| example.....                             | 129 |
| with SLAX statement.....                 | 104 |
| within statement.....                    | 410 |
| usage guidelines.....                    | 347 |

## X

|                                                  |     |
|--------------------------------------------------|-----|
| XML                                              |     |
| advantages of.....                               | 11  |
| overview.....                                    | 10  |
| tags <i>See</i> JUNOS XML tags, JUNOScript tags  |     |
| <xnm:error> JUNOS XML tag.....                   | 273 |
| usage guidelines.....                            | 124 |
| <xnm:warning> JUNOS XML tag.....                 | 275 |
| usage guidelines.....                            | 124 |
| XPath                                            |     |
| function summaries.....                          | 60  |
| overview.....                                    | 17  |
| <xsl:apply-templates> XSLT element.....          | 64  |
| <xsl:call-template> XSLT element.....            | 65  |
| <xsl:choose> XSLT element.....                   | 65  |
| <xsl:choose> XSLT programming instruction.....   | 23  |
| <xsl:comment> XSLT element.....                  | 66  |
| <xsl:copy-of> XSLT element.....                  | 66  |
| <xsl:element> XSLT element.....                  | 66  |
| <xsl:for-each> XSLT element.....                 | 67  |
| <xsl:for-each> XSLT programming instruction..... | 23  |
| <xsl:if> XSLT element.....                       | 67  |
| <xsl:if> XSLT programming instruction.....       | 23  |
| <xsl:import> XSLT element.....                   | 68  |
| <xsl:otherwise> XSLT element.....                | 68  |
| <xsl:param> XSLT element.....                    | 69  |
| <xsl:stylesheet> XSLT element.....               | 70  |
| <xsl:template> XSLT element.....                 | 71  |
| <xsl:text> XSLT element.....                     | 72  |
| <xsl:value-of> XSLT element.....                 | 73  |
| <xsl:variable> XSLT element.....                 | 73  |
| <xsl:when> XSLT element.....                     | 74  |

|                                     |     |
|-------------------------------------|-----|
| <xsl:with-param> XSLT element.....  | 75  |
| XSLT                                |     |
| attribute summaries.....            | 64  |
| comments.....                       | 36  |
| context node.....                   | 25  |
| converting to SLAX                  |     |
| commit scripts.....                 | 175 |
| event scripts.....                  | 428 |
| op scripts.....                     | 304 |
| overview.....                       | 29  |
| dot node.....                       | 25  |
| element summaries.....              | 64  |
| extension functions.....            | 76  |
| summaries.....                      | 76  |
| usage guidelines.....               | 39  |
| <i>See also</i> XSLT functions      |     |
| flow of operation illustrated.....  | 16  |
| function summaries.....             | 60  |
| functions <i>See</i> XSLT functions |     |
| named templates.....                | 19  |
| namespace, in SLAX.....             | 37  |
| overview.....                       | 15  |
| parameters.....                     | 20  |
| programming instructions            |     |
| <xsl:choose>.....                   | 23  |
| <xsl:for-each>.....                 | 23  |
| <xsl:if>.....                       | 23  |
| recursion.....                      | 25  |
| templates <i>See</i> XSLT templates |     |
| unnamed templates.....              | 19  |
| variables.....                      | 21  |
| XPath.....                          | 17  |
| xslt (tracing flag)                 |     |
| commit scripts.....                 | 180 |
| event scripts.....                  | 432 |
| op scripts.....                     | 308 |
| XSLT elements                       |     |
| <xsl:apply-templates>.....          | 64  |
| <xsl:call-template>.....            | 65  |
| <xsl:choose>.....                   | 65  |
| <xsl:comment>.....                  | 66  |
| <xsl:copy-of>.....                  | 66  |
| <xsl:element>.....                  | 66  |
| <xsl:for-each>.....                 | 67  |
| <xsl:if>.....                       | 67  |
| <xsl:import>.....                   | 68  |
| <xsl:otherwise>.....                | 68  |
| <xsl:param>.....                    | 69  |
| <xsl:stylesheet>.....               | 70  |
| <xsl:template>.....                 | 71  |
| <xsl:text>.....                     | 72  |
| <xsl:value-of>.....                 | 73  |
| <xsl:variable>.....                 | 73  |
| <xsl:when>.....                     | 74  |
| <xsl:with-param>.....               | 75  |

## XSLT functions

|                    |    |
|--------------------|----|
| concat()           | 60 |
| contains()         | 60 |
| count()            | 60 |
| jcs:break-lines()  | 76 |
| usage guidelines   | 40 |
| jcs:close()        | 76 |
| usage guidelines   | 40 |
| jcs:dampen()       | 77 |
| usage guidelines   | 41 |
| jcs:empty()        | 77 |
| usage guidelines   | 41 |
| jcs:execute()      | 78 |
| usage guidelines   | 42 |
| jcs:first-of()     | 78 |
| usage guidelines   | 42 |
| jcs:get-input()    | 78 |
| usage guidelines   | 43 |
| jcs:get-secret()   | 79 |
| usage guidelines   | 43 |
| jcs:hostname()     | 79 |
| usage guidelines   | 43 |
| jcs:invoke()       | 80 |
| usage guidelines   | 44 |
| jcs:open()         | 80 |
| usage guidelines   | 44 |
| jcs:output()       | 81 |
| usage guidelines   | 45 |
| jcs:parse-ip()     | 82 |
| usage guidelines   | 45 |
| jcs:printf()       | 83 |
| usage guidelines   | 46 |
| jcs:progress()     | 83 |
| usage guidelines   | 46 |
| jcs:regex()        | 84 |
| usage guidelines   | 46 |
| jcs:sleep()        | 84 |
| usage guidelines   | 47 |
| jcs:split()        | 85 |
| usage guidelines   | 47 |
| jcs:sysctl()       | 85 |
| usage guidelines   | 48 |
| jcs:syslog()       | 86 |
| usage guidelines   | 49 |
| jcs:trace()        | 86 |
| usage guidelines   | 50 |
| last()             | 61 |
| name()             | 61 |
| not()              | 61 |
| position()         | 62 |
| starts-with()      | 62 |
| string-length()    | 62 |
| substring-after()  | 63 |
| substring-before() | 63 |

## XSLT templates

|                     |    |
|---------------------|----|
| <jcs:edit-path >    | 87 |
| usage guidelines    | 51 |
| <jcs:emit-change >  | 88 |
| usage guidelines    | 52 |
| <jcs:emit-comment > | 89 |
| usage guidelines    | 54 |
| <jcs:statement >    | 89 |
| usage guidelines    | 55 |
| overview            | 18 |
| summaries           | 87 |

# Index of Statements and Commands

## A

|                                     |     |
|-------------------------------------|-----|
| allow-transients statement.....     | 277 |
| apply-macro statement .....         | 278 |
| apply-templates SLAX statement..... | 91  |
| archive-sites statement.....        | 383 |
| arguments statement                 |     |
| event policy.....                   | 384 |
| op scripts.....                     | 333 |
| attributes-match statement.....     | 384 |

## C

|                              |     |
|------------------------------|-----|
| call SLAX statement.....     | 92  |
| < change > XSLT element..... | 270 |
| command statement.....       | 334 |
| commands statement.....      | 385 |
| commit statement.....        | 279 |

## D

|                              |     |
|------------------------------|-----|
| description statement        |     |
| op script arguments.....     | 334 |
| op scripts.....              | 334 |
| destination statement        |     |
| event policy.....            | 409 |
| destinations statement.....  | 387 |
| direct-access statement..... | 279 |

## E

|                                      |     |
|--------------------------------------|-----|
| else if SLAX statement.....          | 94  |
| else SLAX statement.....             | 94  |
| equals statement.....                | 387 |
| event-options statement.....         | 388 |
| event-script statement               |     |
| defining script.....                 | 437 |
| invoking script in event policy..... | 390 |
| events statement.....                | 391 |
| execute-commands statement.....      | 393 |

## F

|                              |     |
|------------------------------|-----|
| file statement               |     |
| commit scripts.....          | 280 |
| event scripts.....           | 438 |
| op scripts.....              | 335 |
| filename statement           |     |
| event policy.....            | 409 |
| for-each SLAX statement..... | 95  |

## G

|                               |     |
|-------------------------------|-----|
| generate-event statement..... | 394 |
|-------------------------------|-----|

## I

|                        |     |
|------------------------|-----|
| if SLAX statement..... | 96  |
| ignore statement.....  | 394 |

## M

|                           |     |
|---------------------------|-----|
| match SLAX statement..... | 97  |
| matches statement.....    | 395 |
| mode SLAX statement.....  | 98  |

## N

|                    |     |
|--------------------|-----|
| not statement..... | 396 |
|--------------------|-----|

## O

|                                |     |
|--------------------------------|-----|
| op statement.....              | 336 |
| optional statement.....        | 280 |
| output-filename statement..... | 396 |
| output-format statement.....   | 397 |

## P

|                              |     |
|------------------------------|-----|
| param SLAX statement.....    | 99  |
| policy statement.....        | 398 |
| priority SLAX statement..... | 100 |

## R

|                           |     |
|---------------------------|-----|
| raise-trap statement..... | 400 |
|---------------------------|-----|

|                                 |     |
|---------------------------------|-----|
| refresh statement               |     |
| commit scripts.....             | 281 |
| event scripts.....              | 439 |
| op scripts.....                 | 337 |
| refresh-from statement          |     |
| commit scripts.....             | 281 |
| event scripts.....              | 439 |
| op scripts.....                 | 337 |
| remote-execution statement..... | 440 |
| retry-count statement.....      | 400 |
| retry-interval statement.....   | 400 |

**S**

|                               |     |
|-------------------------------|-----|
| scripts statement.....        | 282 |
| source statement              |     |
| commit scripts.....           | 283 |
| event scripts.....            | 441 |
| op scripts.....               | 338 |
| starts-with statement.....    | 401 |
| < syslog > JUNOS XML tag..... | 271 |

**T**

|                                        |     |
|----------------------------------------|-----|
| template SLAX statement.....           | 101 |
| then statement.....                    | 402 |
| time-interval statement.....           | 404 |
| time-of-day statement.....             | 404 |
| traceoptions statement                 |     |
| commit scripts.....                    | 284 |
| event policy.....                      | 405 |
| event scripts.....                     | 442 |
| op scripts.....                        | 284 |
| transfer-delay statement.....          | 407 |
| < transient-change > XSLT element..... | 272 |
| trigger statement.....                 | 408 |

**U**

|                          |     |
|--------------------------|-----|
| upload statement.....    | 409 |
| user-name statement..... | 410 |

**V**

|                             |     |
|-----------------------------|-----|
| var SLAX statement.....     | 102 |
| version SLAX statement..... | 103 |

**W**

|                          |     |
|--------------------------|-----|
| with SLAX statement..... | 104 |
| within statement.....    | 410 |

**X**

|                                    |     |
|------------------------------------|-----|
| < xnm:error > JUNOS XML tag.....   | 273 |
| < xnm:warning > JUNOS XML tag..... | 275 |