



Junos[®] OS

NETCONF XML Management Protocol Developer Guide

Release
15.1



Modified: 2015-11-19

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, Junos, Steel-Belted Radius, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo, the Junos logo, and JunosE are trademarks 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.

Junos[®] OS NETCONF XML Management Protocol Developer Guide

15.1

Copyright © 2015, Juniper Networks, Inc.

All rights reserved.

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

	About the Documentation	xiii
	Documentation and Release Notes	xiii
	Supported Platforms	xiii
	Using the Examples in This Manual	xiv
	Merging a Full Example	xiv
	Merging a Snippet	xv
	Documentation Conventions	xv
	Documentation Feedback	xvii
	Requesting Technical Support	xviii
	Self-Help Online Tools and Resources	xviii
	Opening a Case with JTAC	xviii
Part 1	Overview	
Chapter 1	NETCONF XML Management Protocol Overview	3
	NETCONF XML Management Protocol and Junos XML API Overview	3
	Advantages of Using the NETCONF XML Management Protocol and Junos XML API	4
Chapter 2	NETCONF and Junos XML Tags	7
	XML and Junos OS Overview	7
	XML Overview	9
	Junos XML and NETCONF XML Management Protocol Tag Elements	9
	Document Type Definition	10
	XML and NETCONF XML Management Protocol Conventions Overview	10
	Request and Response Tag Elements	10
	Child Tag Elements of a Request Tag Element	11
	Child Tag Elements of a Response Tag Element	12
	Spaces, Newline Characters, and Other White Space	12
	XML Comments	12
	Predefined Entity References	13
	Mapping Junos OS Commands to Junos XML Tag Elements	14
	Mapping for Command Options with Variable Values	15
	Mapping for Fixed-Form Command Options	15
	Mapping Configuration Statements to Junos XML Tag Elements	15
	Mapping for Hierarchy Levels and Container Statements	16
	Mapping for Objects That Have an Identifier	17
	Mapping for Single-Value and Fixed-Form Leaf Statements	18
	Mapping for Leaf Statements with Multiple Values	19
	Mapping for Multiple Options on One or More Lines	19
	Mapping for Comments About Configuration Statements	20

	Using NETCONF Configuration Response Tag Elements in NETCONF Requests and Configuration Changes	21
Chapter 3	NETCONF Session	23
	NETCONF Session Overview	23
	Understanding the Client Application's Role in a NETCONF Session	24
	Generating Well-Formed XML Documents	24
	Understanding the Request Procedure in a NETCONF Session	26
Chapter 4	NETCONF Perl Client	27
	Understanding the NETCONF Perl Distribution and Sample Scripts	27
Chapter 5	YANG	29
	Understanding YANG on Devices Running Junos OS	29
	YANG Modules Overview	30
Part 2	Installation	
Chapter 6	NETCONF Perl Client	35
	Downloading the NETCONF Perl Client and Prerequisites Package	35
	Installing the NETCONF Perl Client and Prerequisites Package	36
	Verifying the Installation and Version of Perl	36
	Extracting the NETCONF Perl Client and Sample Scripts	36
	Extracting and Installing the NETCONF Perl Client Prerequisites Package	37
	Installing the NETCONF Perl Client	39
Chapter 7	NETCONF Java Toolkit	41
	Downloading and Installing the NETCONF Java Toolkit	41
	Downloading the NETCONF Java Toolkit	41
	Installing the NETCONF Java Toolkit	41
	Satisfying Requirements for SSHv2 Connections	41
Part 3	Configuration	
Chapter 8	NETCONF Tracing Operations	45
	NETCONF Tracing Operations Overview	45
	Example: Configuring NETCONF Tracing Operations	46
Chapter 9	NETCONF Session	51
	Establishing an SSH Connection for a NETCONF Session	51
	Establishing an SSH Connection for a NETCONF Session	51
	Prerequisites for Establishing an SSH Connection for NETCONF Sessions	52
	Installing SSH Software on the Configuration Management Server	52
	Configuring a User Account for the Client Application on Devices Running Junos OS	52
	Configuring a Public/Private Key Pair or Password for the Junos OS User Account	53
	Accessing the Keys or Password with the Client Application	54

Enabling NETCONF Service over SSH	55
Prerequisites for Establishing an Outbound SSH Connection for NETCONF Sessions	56
Configuring the Device Running Junos OS for Outbound SSH	56
Installing SSH Software on the Client	58
Receiving and Managing the Outbound SSH Initiation Sequence on the Client	59
Enabling NETCONF Service over SSH	59
Connecting to the NETCONF Server	60
Starting the NETCONF Session	61
Exchanging <hello> Tag Elements	61
Verifying Compatibility	63
Sending a Request to the NETCONF Server	65
Operational Requests	65
Configuration Information Requests	66
Configuration Change Requests	67
Parsing the NETCONF Server Response	67
Operational Responses	68
Configuration Information Responses	69
Configuration Change Responses	69
Using a Standard API to Parse Response Tag Elements in NETCONF and Junos XML Protocol Sessions	70
Handling an Error or Warning in a NETCONF Session	70
Locking and Unlocking the Candidate Configuration Using NETCONF	72
Locking the Candidate Configuration	72
Unlocking the Candidate Configuration	73
Terminating a NETCONF Session	74
Ending a NETCONF Session and Closing the Connection	75
Sample NETCONF Session	76
Exchanging Initialization Tag Elements	76
Sending an Operational Request	77
Locking the Configuration	77
Changing the Configuration	78
Committing the Configuration	78
Unlocking the Configuration	78
Closing the NETCONF Session	79
Configuring RFC-Compliant NETCONF Sessions	79
Namespaces	80
Changes to <get> and <get-config> Operations	81
Chapter 10 Changing the Configuration	83
Editing the Candidate Configuration Using NETCONF	83
Uploading and Formatting Configuration Data in a NETCONF Session	85
Referencing Configuration Data Files	85
Streaming Configuration Data	87
Formatting Data: Junos XML versus CLI Configuration Statements	88
Setting the Edit Configuration Mode in a NETCONF Session	90
Specifying the merge Data Mode	91
Specifying the replace Data Mode	91

	Specifying the no-change Data Mode	92
	Handling Errors While Editing the Candidate Configuration in a NETCONF Session	93
	Replacing the Candidate Configuration Using NETCONF	94
	Using <copy-config> to Replace the Candidate Configuration	94
	Using <edit-config> to Replace the Candidate Configuration	94
	Rolling Back the Candidate Configuration Using NETCONF	95
	Deleting the Candidate Configuration Using NETCONF	96
	Changing Individual Configuration Elements Using NETCONF	96
	Merging Configuration Elements Using NETCONF	98
	Creating Configuration Elements Using NETCONF	99
	Deleting Configuration Elements Using NETCONF	101
	Deleting a Hierarchy Level or Container Object	102
	Deleting a Configuration Object That Has an Identifier	102
	Deleting a Single-Value or Fixed-Form Option from a Configuration Object	103
	Deleting Values from a Multi-value Option of a Configuration Object	104
	Replacing Configuration Elements Using NETCONF	106
	Replacing Patterns in Configuration Data Using the NETCONF or Junos XML Protocol	108
	Replacing Patterns Globally Within the Configuration	108
	Replacing Patterns Within a Hierarchy Level or Container Object That Has No Identifier	109
	Replacing Patterns for a Configuration Object That Has an Identifier	110
Chapter 11	Committing the Configuration	113
	Verifying the Configuration Syntax Using NETCONF	113
	Committing the Candidate Configuration Using NETCONF	114
	Committing the Candidate Configuration Only After Confirmation Using NETCONF	114
Chapter 12	Configuration Statements	117
	connection-limit	118
	netconf	119
	port (NETCONF)	120
	rate-limit	121
	rfc-compliant (NETCONF)	122
	ssh (NETCONF)	123
	traceoptions (NETCONF)	124
Chapter 13	NETCONF Protocol Operations	127
	<close-session/>	127
	<commit>	127
	<copy-config>	128
	<delete-config>	129
	<discard-changes/>	129
	<edit-config>	129
	<get>	132
	<get-config>	133
	<kill-session>	134

	<lock>	134
	<unlock>	135
	<validate>	135
Chapter 14	NETCONF Request and Response Tags	137
]]>]]>	137
	<data>	137
	<error-info>	138
	<hello>	138
	<ok/>	139
	<rpc>	140
	<rpc-error>	140
	<rpc-reply>	141
	<target>	141
Chapter 15	Junos XML Protocol Elements Supported in NETCONF Sessions	143
	<abort/>	143
	<abort-acknowledgement/>	143
	<checksum-information>	144
	<close-configuration/>	144
	<commit-configuration>	145
	<commit-results>	149
	<database-status>	150
	<database-status-information>	151
	<end-session/>	151
	<get-checksum-information>	152
	<get-configuration>	152
	<load-configuration>	156
	<load-configuration-results>	159
	<lock-configuration/>	160
	<open-configuration>	161
	<reason>	161
	<request-end-session/>	162
	<routing-engine>	162
	<unlock-configuration/>	163
	<xnm:error>	164
	<xnm:warning>	165
Chapter 16	Junos XML Protocol Element Attributes Supported in NETCONF Sessions	167
	junos:changed-localtime	167
	junos:changed-seconds	167
	junos:commit-localtime	168
	junos:commit-seconds	168
	junos:commit-user	169
	operation	169
	replace-pattern	170
	xmlns	171

Part 4	Administration	
Chapter 17	Requesting Operational Information	175
	Requesting Operational Information Using NETCONF	175
	Specifying the Output Format for Operational Information Requests in a NETCONF Session	177
Chapter 18	Requesting Configuration Information	183
	Requesting the Committed Configuration and Device State Using NETCONF . .	183
	Requesting Configuration Data Using NETCONF	184
	Specifying the Source for Configuration Information Requests Using NETCONF	186
	Specifying the Scope of Configuration Information to Return in a NETCONF Response	188
	Requesting the Complete Configuration Using NETCONF	188
	Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF	189
	Requesting All Configuration Objects of a Specified Type Using NETCONF	191
	Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF	194
	Requesting A Specific Configuration Object Using NETCONF	197
	Requesting Specific Child Tags for a Configuration Object Using NETCONF	199
	Requesting Multiple Configuration Elements Simultaneously Using NETCONF	203
	Requesting a Previous (Rollback) Configuration Using NETCONF	204
	Comparing Two Previous (Rollback) Configurations Using NETCONF	206
	Requesting the Rescue Configuration Using NETCONF	208
	Requesting an XML Schema for the Configuration Hierarchy Using NETCONF . .	210
	Requesting an XML Schema for the Configuration Hierarchy	210
	Creating the junos.xsd File	211
	Example: Requesting an XML Schema	211
Chapter 19	NETCONF Perl Client Applications	215
	Writing NETCONF Perl Client Applications	215
	Importing Perl Modules and Declaring Constants in NETCONF Perl Client Applications	216
	Connecting to the NETCONF Server in Perl Client Applications	217
	Satisfy Protocol Prerequisites	217
	Group Requests	217
	Obtain and Record Parameters Required by the NET::Netconf::Manager Object	217
	Obtaining Application-Specific Parameters	218
	Establishing the Connection	219
	Collecting Parameters Interactively in NETCONF Perl Client Applications	219
	Submitting a Request to the NETCONF Server in Perl Client Applications	222
	Providing Method Options or Attributes	222
	Submitting a Request	224
	Example: Requesting an Inventory of Hardware Components Using a NETCONF Perl Client Application	226

	Example: Changing the Configuration Using a NETCONF Perl Client	
	Application	227
	Handling Error Conditions	227
	Locking the Configuration	228
	Reading In the Configuration Data	228
	Editing the Configuration Data	229
	Committing the Configuration	230
	Parsing and Formatting the Response from the NETCONF Server in Perl Client	
	Applications	230
	Closing the Connection to the NETCONF Server in Perl Client Applications	233
Chapter 20	YANG	235
	Using Juniper Networks YANG Modules	235
	Obtaining Juniper Networks YANG Modules	236
	Importing Juniper Networks YANG Modules	236
	show system schema	238
Part 5	Index	
	Index	241

List of Tables

	About the Documentation	xiii
	Table 1: Notice Icons	xvi
	Table 2: Text and Syntax Conventions	xvi
Part 1	Overview	
Chapter 2	NETCONF and Junos XML Tags	7
	Table 3: Predefined Entity Reference Substitutions for Tag Content Values	13
	Table 4: Predefined Entity Reference Substitutions for Attribute Values	13
Part 4	Administration	
Chapter 20	YANG	235
	Table 5: Juniper Networks YANG Modules	235

About the Documentation

- [Documentation and Release Notes on page xiii](#)
- [Supported Platforms on page xiii](#)
- [Using the Examples in This Manual on page xiv](#)
- [Documentation Conventions on page xv](#)
- [Documentation Feedback on page xvii](#)
- [Requesting Technical Support on page xviii](#)

Documentation and 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/>.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at <http://www.juniper.net/books>.

Supported Platforms

For the features described in this document, the following platforms are supported:

- [ACX Series](#)
- [EX Series](#)
- [M Series](#)
- [MX Series](#)
- [QFabric System](#)
- [QFX Series standalone switches](#)
- [SRX Series](#)
- [T Series](#)

Using the Examples in This Manual

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

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

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

Merging a Full Example

To merge a full example, follow these steps:

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

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

```
system {
  scripts {
    commit {
      file ex-script.xml;
    }
  }
}
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.xml; }
```

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

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

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

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

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

Documentation Conventions

[Table 1 on page xvi](#) 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.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

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

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: 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 or emphasizes important new terms. Identifies guide 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 OS CLI User 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>

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

Convention	Description	Examples
Text like this	Represents names of configuration statements, commands, files, and directories; 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)	Encloses optional keywords or variables.	stub <default-metric <i>metric</i>>;
(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 (<i>string1</i> <i>string2</i> <i>string3</i>)
# (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)	Encloses a variable for which you can substitute one or more values.	community name members [<i>community-ids</i>]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	<pre>[edit] routing-options { static { route default { nexthop <i>address</i>; retain; } } }</pre>
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		
Bold text like this	Represents 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 menu 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 provide feedback by using either of the following methods:

- Online feedback rating system—On any page at the Juniper Networks Technical Documentation site at <http://www.juniper.net/techpubs/index.html>, simply click the stars to rate the content, and use the pop-up form to provide us with information about your experience. Alternately, you can use the online feedback form at <http://www.juniper.net/techpubs/feedback/>.

- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software 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 Partner Support Service support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

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

Self-Help Online Tools and Resources

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

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

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

Opening a Case with JTAC

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

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

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

PART 1

Overview

- [NETCONF XML Management Protocol Overview on page 3](#)
- [NETCONF and Junos XML Tags on page 7](#)
- [NETCONF Session on page 23](#)
- [NETCONF Perl Client on page 27](#)
- [YANG on page 29](#)

CHAPTER 1

NETCONF XML Management Protocol Overview

- [NETCONF XML Management Protocol and Junos XML API Overview on page 3](#)
- [Advantages of Using the NETCONF XML Management Protocol and Junos XML API on page 4](#)

NETCONF XML Management Protocol and Junos XML API Overview

The NETCONF XML management protocol is an XML-based protocol that client applications use to request and change configuration information on routing, switching, and security devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data and remote procedure calls. The NETCONF protocol defines basic operations that are equivalent to configuration mode commands in the command-line interface (CLI). Applications use the protocol operations to display, edit, and commit configuration statements (among other operations), just as administrators use CLI configuration mode commands to perform those operations.

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

The NETCONF XML management protocol is described in RFC 4741, *NETCONF Configuration Protocol*, which is available at <http://www.ietf.org/rfc/rfc4741.txt>.

Client applications request information and change the configuration on a switch, router, or security device by encoding the request with tag elements from the NETCONF XML management protocol and Junos XML API and then sending it to the NETCONF server on the device. On devices running Junos OS, the NETCONF server is integrated into Junos OS and does not appear as a separate entry in process listings. The NETCONF server directs the request to the appropriate software modules within the device, encodes the response in NETCONF and Junos XML API tag elements, and returns the result to the client application. For example, to request information about the status of a device's interfaces, a client application sends the Junos XML API **<get-interface-information>**

request. The NETCONF server gathers the information from the interface process and returns it in the Junos XML API **<interface-information>** response tag element.

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

**Related
Documentation**

- [Advantages of Using the NETCONF XML Management Protocol and Junos XML API on page 4](#)
- [XML and Junos OS Overview on page 7](#)
- [XML Overview on page 9](#)

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

The NETCONF XML management protocol and Junos XML API 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. NETCONF and Junos XML tag elements make it straightforward for client applications that request information from a device to parse the output and find specific information.

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

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

The corresponding XML-tagged version is:

```
<interface>
  <name>fxp0</name>
  <admin-status>enabled</admin-status>
  <operational-status>up</operational-status>
  <index>4</index>
  <snmp-index>3</snmp-index>
</interface>
```


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

, SNMP ifIndex

A problem arises if the format or ordering of output changes in a later version of the Junos OS, 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 NETCONF 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 device component at different times. When a device 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.

CHAPTER 2

NETCONF and Junos XML Tags

- [XML and Junos OS Overview on page 7](#)
- [XML Overview on page 9](#)
- [XML and NETCONF XML Management Protocol Conventions Overview on page 10](#)
- [Mapping Junos OS Commands to Junos XML Tag Elements on page 14](#)
- [Mapping Configuration Statements to Junos XML Tag Elements on page 15](#)
- [Using NETCONF Configuration Response Tag Elements in NETCONF Requests and Configuration Changes on page 21](#)

XML and Junos OS Overview

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

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

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

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

protocol operations. The following example compares the text and XML output for the **show chassis alarms** operational mode command:

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

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

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

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

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

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

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

- Related Documentation**
- *Junos XML API Configuration Developer Reference*
 - *Junos XML API Operational Developer Reference*

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 Junos XML and NETCONF XML management protocol tag elements:

- [Junos XML and NETCONF XML Management Protocol Tag Elements on page 9](#)
- [Document Type Definition on page 10](#)

Junos XML and NETCONF XML Management Protocol 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 Junos XML tag element or NETCONF 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.

NETCONF 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>** 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>** element indicates that its contents specify the number of bytes received.

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

XML and NETCONF XML Management Protocol Conventions Overview

A client application must comply with XML and NETCONF XML management protocol conventions. Each request from the client application must be a *well-formed* XML document; that is, it must obey the structural rules defined in the NETCONF and Junos XML document type definitions (DTD)s for the kind of information encoded in the request. The client application must emit tag elements in the required order and only in the legal contexts. Compliant applications are easier to maintain in the event of changes to the Junos OS or NETCONF protocol.

Similarly, each response from the NETCONF server constitutes a well-formed XML document (the NETCONF server obeys XML and NETCONF conventions).

The following sections describe NETCONF XML management protocol conventions:

- [Request and Response Tag Elements on page 10](#)
- [Child Tag Elements of a Request Tag Element on page 11](#)
- [Child Tag Elements of a Response Tag Element on page 12](#)
- [Spaces, Newline Characters, and Other White Space on page 12](#)
- [XML Comments on page 12](#)
- [Predefined Entity References on page 13](#)

Request and Response Tag Elements

A *request* tag element is one generated by a client application to request information about a device's current status or configuration, or to change the configuration. A request tag element corresponds to a CLI operational or configuration command. It can occur only within an **<rpc>** tag. For information about the **<rpc>** element, see [“Sending a Request to the NETCONF Server” on page 65](#).

A *response* tag element represents the NETCONF server's reply to a request tag element and occurs only within an `<rpc-reply>` tag. For information about the `<rpc-reply>` element, see [“Parsing the NETCONF Server Response” on page 67](#).

The following example represents an exchange in which a client application emits the `<get-interface-information>` request tag element with the `<extensive/>` flag and the NETCONF server returns the `<interface-information>` response tag element.

Client Application	<pre> <rpc> <get-interface-information> <extensive/> </get-interface-information> </rpc>]]>]]> </pre>
NETCONF Server	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <interface-information xmlns="URL"> <!-- children of <interface-information> --> </interface-information> </rpc-reply>]]>]]> </pre>



NOTE: This example, like all others in this guide, shows each tag element on a separate line, in the tag streams emitted by both the client application and NETCONF server. In practice, a client application does not need to include newline characters between tag elements, because the server automatically discards such white space. For further discussion, see [“Spaces, Newline Characters, and Other White Space” on page 12](#).

For information about the attributes in the opening `<rpc-reply>` tag, see [“Parsing the NETCONF Server Response” on page 67](#). For information about the `xmlns` attribute in the opening `<interface-information>` tag, see [“Requesting Operational Information Using NETCONF” on page 175](#). For information about the `]]>]]>` character sequence, see [“Generating Well-Formed XML Documents” on page 24](#).

Child Tag Elements of a Request Tag Element

Some request tag elements contain child tag elements. For configuration requests, each child tag element represents a configuration element (hierarchy level or configuration object). For operational requests, each child tag element represents one of the options you provide on the command line when issuing the equivalent CLI command.

Some requests have mandatory child tag elements. To make a request successfully, a client application must emit the mandatory tag elements within the request tag element's opening and closing tags. If any of the children are themselves container tag elements, the opening tag for each must occur before any of the tag elements it contains, and the closing tag must occur before the opening tag for another tag element at its hierarchy level.

In most cases, the client application can emit children that occur at the same level within a container tag element in any order. The important exception is a configuration element

that has an *identifier tag element*, which distinguishes the configuration element from other elements of its type. The identifier tag element must be the first child tag element in the container tag element. Most frequently, the identifier tag element specifies the name of the configuration element and is called **<name>**. For more information, see [“Mapping for Objects That Have an Identifier” on page 15](#).

Child Tag Elements of a Response Tag Element

The child tag elements of a response tag element represent the individual data items returned by the NETCONF server for a particular request. The children can be either individual tag elements (empty tags or tag element triples) or container tag elements that enclose their own child tag elements. For some container tag elements, the NETCONF server returns the children in alphabetical order. For other elements, the children appear in the order in which they were created in the configuration.

The set of child tag elements that can occur in a response or within a container tag element is subject to change in later releases of the Junos XML API. Client applications must not rely on the presence or absence of a particular tag element in the NETCONF server’s output, nor on the ordering of child tag elements within a response tag element. For the most robust operation, include logic in the client application that handles the absence of expected tag elements or the presence of unexpected ones as gracefully as possible.

Spaces, Newline Characters, and Other White Space

As dictated by the XML specification, the NETCONF server ignores white space (spaces, tabs, newline characters, and other characters that represent white space) that occurs between tag elements in the tag stream generated by a client application. Client applications can, but do not need to, include white space between tag elements. However, they must not insert white space within an opening or closing tag. If they include white space in the contents of a tag element that they are submitting as a change to the candidate configuration, the NETCONF server preserves the white space in the configuration database.

In its responses, the NETCONF server includes white space between tag elements to enhance the readability of responses that are saved to a file: it uses newline characters to put each tag element on its own line, and spaces to indent child tag elements to the right compared to their parents. A client application can ignore or discard the white space, particularly if it does not store responses for later review by human users. However, it must not depend on the presence or absence of white space in any particular location when parsing the tag stream.

For more information about white space in XML documents, see the XML specification from the World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0*, at <http://www.w3.org/TR/REC-xml/>.

XML Comments

Client applications and the NETCONF server can insert XML comments at any point between tag elements in the tag stream they generate, but not within tag elements. Client applications must handle comments in output from the NETCONF server gracefully

but must not depend on their content. Client applications also cannot use comments to convey information to the NETCONF server, because the server automatically discards any comments it receives.

XML comments are enclosed within the strings `<!--` and `-->`, and cannot contain the string `--` (two hyphens). For more details about comments, see the XML specification at <http://www.w3.org/TR/REC-xml/>.

The following is an example of an XML comment:

```
<!-- This is a comment. Please ignore it. -->
```

Predefined Entity References

By XML convention, there are two contexts in which certain characters cannot appear in their regular form:

- In the string that appears between opening and closing tags (the contents of the tag element)
- In the string value assigned to an attribute of an opening tag

When including a disallowed character in either context, client applications must substitute the equivalent *predefined entity reference*, which is a string of characters that represents the disallowed character. Because the NETCONF server uses the same predefined entity references in its response tag elements, the client application must be able to convert them to actual characters when processing response tag elements.

[Table 3 on page 13](#) summarizes the mapping between disallowed characters and predefined entity references for strings that appear between the opening and closing tags of a tag element.

Table 3: Predefined Entity Reference Substitutions for Tag Content Values

Disallowed Character	Predefined Entity Reference
& (ampersand)	&
> (greater-than sign)	>
< (less-than sign)	<

[Table 4 on page 13](#) summarizes the mapping between disallowed characters and predefined entity references for attribute values.

Table 4: Predefined Entity Reference Substitutions for Attribute Values

Disallowed Character	Predefined Entity Reference
& (ampersand)	&

Table 4: Predefined Entity Reference Substitutions for Attribute Values (*continued*)

Disallowed Character	Predefined Entity Reference
' (apostrophe)	'
> (greater-than sign)	>
< (less-than sign)	<
" (quotation mark)	"

As an example, suppose that the following string is the value contained by the **<condition>** tag element:

```
if (a<b && b>c) return "Peer's not responding"
```

The **<condition>** tag element looks like this (it appears on two lines for legibility only):

```
<condition>if (a&lt;b &amp;&amp; b&gt;c) return "Peer's not \
responding"</condition>
```

Similarly, if the value for the **<example>** tag element's **heading** attribute is **Peer's "age" <> 40**, the opening tag looks like this:

```
<example heading="Peer&apos;s &quot;age&quot; &lt;&gt; 40">
```

Mapping Junos OS Commands to Junos XML Tag Elements

The Junos XML API is an XML representation of Junos OS configuration statements and operational mode commands. It defines an XML equivalent for all statements in the Junos OS configuration hierarchy and many of the commands that you issue in CLI operational mode. Each operational mode command with a Junos XML counterpart maps to a request tag element and, if necessary, a response tag element. Request tag elements are used in remote procedure calls (RPCs) within Junos XML protocol sessions to request information from a device running Junos OS. For example, the **show interfaces** command maps to the **<get-interface-information>** request tag.

Information about the available command equivalents in the current release of the Junos OS can be found in the *Junos XML API Operational Developer Reference*. For the mapping between commands and Junos XML tag elements, see the *Junos XML API Operational Developer Reference* "Mapping Between Operational Tag Elements, Perl Methods, and CLI Commands" chapter. For detailed information about a specific operation, see the *Junos XML API Operational Developer Reference* "Summary of Operational Request Tags" chapter.

The following sections describe the tag elements that map to command options:

- [Mapping for Command Options with Variable Values on page 15](#)
- [Mapping for Fixed-Form Command Options on page 15](#)

Mapping for Command Options with Variable Values

Many CLI commands have options that identify the object that the command affects or reports about, distinguishing the object from other objects of the same type. In some cases, the CLI does not precede the identifier with a fixed-form keyword, but XML convention requires that the Junos XML API define a tag element for every option. To learn the names for each identifier (and any other child tag elements) for an operational request tag element, consult the tag element's entry in the appropriate DTD or in the *Junos XML API Operational Developer Reference*.

The following example shows the Junos XML tag elements for two CLI operational commands that have variable-form options. In the **show interfaces** command, t3-5/1/0:0 is the name of the interface. In the **show bgp neighbor** command, 10.168.1.222 is the IP address for the BGP peer of interest.

CLI Command	JUNOS XML Tags
show interfaces t3-5/1/0:0	<pre> <rpc> <get-interface-information> <interface-name>t3-5/1/0:0</interface-name> </get-interface-information> </rpc> </pre>
show bgp neighbor 10.168.1.222	<pre> <rpc> <get-bgp-neighbor-information> <neighbor-address>10.168.1.222</neighbor-address> </get-bgp-neighbor-information> </rpc> </pre>

T1500

Mapping for Fixed-Form Command Options

Some CLI commands include options that have a fixed form, such as the **brief** and **detail** strings, which specify the amount of detail to include in the output. The Junos XML API usually maps such an option to an empty tag whose name matches the option name.

The following example shows the Junos XML tag elements for the **show isis adjacency** command, which has a fixed-form option called **detail**.

CLI Command	JUNOS XML Tags
show isis adjacency detail	<pre> <rpc> <get-isis-adjacency-information> <detail/> </get-isis-adjacency-information> </rpc> </pre>

T1501

Related Documentation

- *Mapping Junos OS Commands and Command Output to Junos XML Tag Elements Using the CLI*

Mapping Configuration Statements to Junos XML Tag Elements

The Junos XML API defines a tag element for every container and leaf statement in the configuration hierarchy. At the top levels of the configuration hierarchy, there is almost always a one-to-one mapping between tag elements and statements, and most tag names match the configuration statement name. At deeper levels of the hierarchy, the

mapping is sometimes less direct, because some CLI notational conventions do not map directly to XML-compliant tagging syntax.



NOTE: For some configuration statements, the notation used when you type the statement at the CLI configuration-mode prompt differs from the notation used in a configuration file. The same Junos XML tag element maps to both notational styles.

The following sections describe the mapping between configuration statements and Junos XML tag elements:

- [Mapping for Hierarchy Levels and Container Statements on page 16](#)
- [Mapping for Objects That Have an Identifier on page 17](#)
- [Mapping for Single-Value and Fixed-Form Leaf Statements on page 18](#)
- [Mapping for Leaf Statements with Multiple Values on page 19](#)
- [Mapping for Multiple Options on One or More Lines on page 19](#)
- [Mapping for Comments About Configuration Statements on page 20](#)

Mapping for Hierarchy Levels and Container Statements

The `<configuration>` element is the top-level Junos XML container element for configuration statements. It corresponds to the **[edit]** hierarchy level in CLI configuration mode. Most statements at the next few levels of the configuration hierarchy are container statements. The Junos XML container tag element that corresponds to a container statement almost always has the same name as the statement.

The following example shows the Junos XML tag elements for two statements at the top level of the configuration hierarchy. Note that a closing brace in a CLI configuration statement corresponds to a closing Junos XML tag.

CLI Configuration Statements	JUNOS XML Tags
system {	<code><configuration></code>
login {	<code><system></code>
...child statements...	<code><login></code>
}	<code><!-- tags for child statements --></code>
}	<code></login></code>
	<code></system></code>
protocols {	<code><protocols></code>
ospf {	<code><ospf></code>
...child statements...	<code><!-- tags for child statements --></code>
}	<code></ospf></code>
}	<code></protocols></code>
	<code></configuration></code>

T1502

Mapping for Objects That Have an Identifier

At some hierarchy levels, the same kind of configuration object can occur multiple times. Each instance of the object has a unique identifier to distinguish it from the other instances. In the CLI notation, the parent statement for such an object consists of a keyword and identifier of the following form:

```
keyword identifier {
... configuration statements for individual characteristics ...
}
```

keyword is a fixed string that indicates the type of object being defined, and **identifier** is the unique name for this instance of the type. In the Junos XML API, the tag element corresponding to the keyword is a container tag element for child tag elements that represent the object's characteristics. The container tag element's name generally matches the **keyword** string.

The Junos XML API differs from the CLI in its treatment of the identifier. Because the Junos XML API does not allow container tag elements to contain both other tag elements and untagged character data such as an identifier name, the identifier must be enclosed in a tag element of its own. Most frequently, identifier tag elements for configuration objects are called **<name>**. Some objects have multiple identifiers, which usually have names other than **<name>**. To verify the name of each identifier tag element for a configuration object, consult the entry for the object in the *Junos XML API Configuration Developer Reference*.



NOTE: The Junos OS reserves the prefix **junos-** for the identifiers of configuration groups defined within the **junos-defaults** configuration group. User-defined identifiers cannot start with the string **junos-**.

Identifier tag elements also constitute an exception to the general XML convention that tag elements at the same level of hierarchy can appear in any order; the identifier tag element always occurs first within the container tag element.

The configuration for most objects that have identifiers includes additional leaf statements, which represent other characteristics of the object. For example, each BGP group configured at the **[edit protocols bgp group]** hierarchy level has an associated name (the identifier) and can have leaf statements for other characteristics such as type, peer autonomous system (AS) number, and neighbor address. For information about the Junos XML mapping for leaf statements, see [“Mapping for Single-Value and Fixed-Form Leaf Statements” on page 18](#), [“Mapping for Leaf Statements with Multiple Values” on page 19](#), and [“Mapping for Multiple Options on One or More Lines” on page 19](#).

The following example shows the Junos XML tag elements for configuration statements that define two BGP groups called **<name>** and **<name>**. Notice that the Junos XML **<name>** element that encloses the identifier of each group (and the identifier of the neighbor within a group) does not have a counterpart in the CLI statements.

CLI Configuration Statements	JUNOS XML Tags
<pre> protocols { bgp { group G1 { type external; peer-as 56; neighbor 10.0.0.1; } group G2 { type external; peer-as 57; neighbor 10.0.10.1; } } } </pre>	<pre> <configuration> <protocols> <bgp> <group> <name>G1</name> <type>external</type> <peer-as>56</peer-as> <neighbor> <name>10.0.0.1</name> </neighbor> </group> <group> <name>G2</name> <type>external</type> <peer-as>57</peer-as> <neighbor> <name>10.0.10.1</name> </neighbor> </group> </bgp> </protocols> </configuration> </pre>

T1503

Mapping for Single-Value and Fixed-Form Leaf Statements

A *leaf statement* is a CLI configuration statement that does not contain any other statements. Most leaf statements define a value for one characteristic of a configuration object and have the following form:

keyword *value*;

In general, the name of the Junos XML tag element corresponding to a leaf statement is the same as the **keyword** string. The string between the opening and closing Junos XML tags is the same as the *value* string.

The following example shows the Junos XML tag elements for two leaf statements that have a keyword and a value: the **message** statement at the **[edit system login]** hierarchy level and the **preference** statement at the **[edit protocols ospf]** hierarchy level.

CLI Configuration Statements	JUNOS XML Tags
<pre> system { login { message "Authorized users only"; ...other statements under login... } } protocols { ospf { preference 15; ...other statements under ospf... } } </pre>	<pre> <configuration> <system> <login> <message>Authorized users only</message> <!-- tags for other child statements --> </login> </system> <protocols> <ospf> <preference>15</preference> <!-- tags for other child statements --> </ospf> </protocols> </configuration> </pre>

T1504

Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. The Junos XML API represents such statements with an empty tag. The following example shows the Junos XML tag elements for the **disable** statement at the **[edit forwarding-options sampling]** hierarchy level.

CLI Configuration Statement	JUNOS XML Tags	
forwarding-options { sampling { disable; ...other statements under sampling ... } }	<configuration> <forwarding-options> <sampling> <disable/> <!-- tags for other child statements --> </sampling> </forwarding-options> </configuration>	T1505

Mapping for Leaf Statements with Multiple Values

Some Junos leaf statements accept multiple values, which can be either user-defined or drawn from a set of predefined values. CLI notation uses square brackets to enclose all values in a single statement, as in the following:

statement [*value1 value2 value3 ...*];

The Junos XML API instead encloses each value in its own tag element. The following example shows the Junos XML tag elements for a CLI statement with multiple user-defined values. The **import** statement imports two routing policies defined elsewhere in the configuration.

CLI Configuration Statements	JUNOS XML Tags	
protocols { bgp { group 23 { import [policy1 policy2]; } } }	<configuration> <protocols> <bgp> <group> <name>23</name> <import>policy1</import> <import>policy2</import> </group> </bgp> </protocols> </configuration>	T1506

The following example shows the Junos XML tag elements for a CLI statement with multiple predefined values. The **permissions** statement grants three predefined permissions to members of the **user-accounts** login class.

CLI Configuration Statements	JUNOS XML Tags	
system { login { class user-accounts { permissions [configure admin control]; } } }	<configuration> <system> <login> <class> <name>user-accounts</name> <permissions>configure</permissions> <permissions>admin</permissions> <permissions>control</permissions> </class> </login> </system> </configuration>	T1507

Mapping for Multiple Options on One or More Lines

For some Junos configuration objects, the standard CLI syntax places multiple options on a single line, usually for greater legibility and conciseness. In most such cases, the first option identifies the object and does not have a keyword, but later options are paired

keywords and values. The Junos XML API encloses each option in its own tag element. Because the first option has no keyword in the CLI statement, the Junos XML API assigns a name to its tag element.

The following example shows the Junos XML tag elements for a CLI configuration statement with multiple options on a single line. The Junos XML API defines a tag element for both options and assigns a name to the tag element for the first option (10.0.0.1), which has no CLI keyword.

CLI Configuration Statements

```
system {
  backup-router 10.0.0.1 destination 10.0.0.2;
}
```

JUNOS XML Tags

```
<configuration>
  <system>
    <backup-router>
      <address>10.0.0.1</address>
      <destination>10.0.0.2</destination>
    </backup-router>
  </system>
</configuration>
```

T1508

The syntax for some configuration objects includes more than one multioption line. Again, the Junos XML API defines a separate tag element for each option. The following example shows Junos XML tag elements for a **traceoptions** statement at the **[edit protocols isis]** hierarchy level. The statement has three child statements, each with multiple options.

CLI Configuration Statements

```
protocols {
  isis {
    traceoptions {
      file trace-file size 3m files 10 world-readable;

      flag route detail;

      flag state receive;
    }
  }
}
```

JUNOS XML Tags

```
<configuration>
  <protocols>
    <isis>
      <traceoptions>
        <file>
          <filename>trace-file</filename>
          <size>3m</size>
          <files>10</files>
          <world-readable/>
        </file>
        <flag>
          <name>route</name>
          <detail/>
        </flag>
        <flag>
          <name>state</name>
          <receive/>
        </flag>
      </traceoptions>
    </isis>
  </protocols>
</configuration>
```

T1509

Mapping for Comments About Configuration Statements

A Junos configuration can include comments that describe statements in the configuration. In CLI configuration mode, the **annotate** command specifies the comment to associate with a statement at the current hierarchy level. You can also use a text editor to insert comments directly into a configuration file. For more information, see the *CLI User Guide*.

The Junos XML API encloses comments about configuration statements in the **<junos:comment>** element. (These comments are different from the comments that are enclosed in the strings **<!--** and **-->** and are automatically discarded by the protocol server.)

In the Junos XML API, the `<junos:comment>` element immediately precedes the element for the associated configuration statement. (If the tag element for the associated statement is omitted, the comment is not recorded in the configuration database.) The comment text string can include one of the two delimiters that indicate a comment in the configuration database: either the `#` character before the comment or the paired strings `/*` before the comment and `*/` after it. If the client application does not include the delimiter, the protocol server adds the appropriate one when it adds the comment to the configuration. The protocol server also preserves any white space included in the comment.

The following example shows the Junos XML tag elements that associate comments with two statements in a sample configuration statement. The first comment illustrates how including newline characters in the contents of the `<junos:comment>` element (`/* New backbone area */`) results in the comment appearing on its own line in the configuration file. There are no newline characters in the contents of the second `<junos:comment>` element, so in the configuration file the comment directly follows the associated statement on the same line.

CLI Configuration Statements	JUNOS XML Tags
protocols {	<code><configuration></code>
ospf {	<code><protocols></code>
/* New backbone area */	<code><ospf></code>
area 0.0.0.0 {	<code><junos:comment></code>
interface so-0/0/0 { # From jnpr1 to jnpr2	<code>/* New backbone area */</code>
hello-interval 5;	<code></junos:comment></code>
}	<code><area></code>
}	<code><name>0.0.0.0</name></code>
}	<code><junos:comment> # From jnpr1 to jnpr2</junos:comment></code>
}	<code><interface></code>
	<code><name>so-0/0/0</name></code>
	<code><hello-interval>5</hello-interval></code>
	<code></interface></code>
	<code></area></code>
	<code></ospf></code>
	<code></protocols></code>
	<code></configuration></code>

T1510

Using NETCONF Configuration Response Tag Elements in NETCONF Requests and Configuration Changes

The NETCONF server encloses its response to each configuration request in `<rpc-reply>` and `<configuration>` tag elements. Enclosing each configuration response within a `<configuration>` tag element contrasts with how the server encloses each different operational response in a tag element named for that type of response—for example, the `<chassis-inventory>` tag element for chassis information or the `<interface-information>` tag element for interface information.

The Junos XML tag elements within the `<configuration>` tag element represent configuration hierarchy levels, configuration objects, and object characteristics, always ordered from higher to deeper levels of the hierarchy. When a client application loads a configuration, it can emit the same tag elements in the same order as the NETCONF server uses when returning configuration information. This consistent representation makes handling configuration information more straightforward. For instance, the client application can request the current configuration, store the NETCONF server's response in a local memory buffer, make changes or apply transformations to the buffered data,

and submit the altered configuration as a change to the candidate configuration. Because the altered configuration is based on the NETCONF server's response, it is certain to be syntactically correct.

Similarly, when a client application requests information about a configuration element (hierarchy level or configuration object), it uses the same tag elements that the NETCONF server will return in response. To represent the element, the client application sends a complete stream of tag elements from the top of the configuration hierarchy (represented by the **<configuration>** tag element) down to the requested element. The innermost tag element, which represents the level or object, is either empty or includes the identifier tag element only. The NETCONF server's response includes the same stream of parent tag elements, but the tag element for the requested configuration element contains all the tag elements that represent the element's characteristics or child levels. For more information, see [“Requesting Configuration Data Using NETCONF” on page 184](#).

The tag streams emitted by the NETCONF server and by a client application can differ in the use of white space, as described in [“XML and NETCONF XML Management Protocol Conventions Overview” on page 10](#).

**Related
Documentation**

- [XML and NETCONF XML Management Protocol Conventions Overview on page 10](#)
- [Mapping Configuration Statements to Junos XML Tag Elements on page 15](#)
- [Requesting Configuration Data Using NETCONF on page 184](#)

CHAPTER 3

NETCONF Session

- [NETCONF Session Overview on page 23](#)
- [Understanding the Client Application's Role in a NETCONF Session on page 24](#)
- [Generating Well-Formed XML Documents on page 24](#)
- [Understanding the Request Procedure in a NETCONF Session on page 26](#)

NETCONF Session Overview

Communication between the NETCONF server and a client application is session based. The server and client explicitly establish a connection and session before exchanging data and close the session and connection when they are finished.

The streams of NETCONF and Junos XML tag elements emitted by the NETCONF server and the client application must each constitute well-formed XML by obeying the structural rules defined in the document type definition (DTD) for the kind of information they are exchanging. The client application must emit tag elements in the required order and only in the allowed contexts.

Client applications access the NETCONF server using the SSH protocol and use the standard SSH authentication mechanism. After authentication, the NETCONF server uses the configured Junos OS login usernames and classes to determine whether a client application is authorized to make each request.

1. The following list outlines the basic structure of a NETCONF session.
2. The client application establishes a connection to the NETCONF server and opens the NETCONF session.
3. The NETCONF server and client application exchange initialization information, which is used to determine if they are using compatible versions of the Junos OS and the NETCONF XML management protocol.
4. The client application sends one or more requests to the NETCONF server and parses its responses.
5. The client application closes the NETCONF session and the connection to the NETCONF server.

For an example of a complete NETCONF session, see [“Sample NETCONF Session” on page 76](#).

- Related Documentation**
- [Generating Well-Formed XML Documents on page 24](#)
 - [Connecting to the NETCONF Server on page 60](#)
 - [Starting the NETCONF Session on page 61](#)
 - [Sample NETCONF Session on page 76](#)

Understanding the Client Application's Role in a NETCONF Session

To create a session and communicate with the NETCONF server, a client application performs the following procedures, which are described in the indicated sections:

1. Satisfies the prerequisites for an SSH connection, as described in [“Establishing an SSH Connection for a NETCONF Session” on page 51](#).
2. Establishes a connection to the NETCONF server on the routing platform, as described in [“Connecting to the NETCONF Server” on page 60](#).
3. Opens a NETCONF session, as described in [“Starting the NETCONF Session” on page 61](#).
4. (Optional) Locks the candidate configuration, as described in [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#). Locking the configuration prevents other users or applications from changing it at the same time.
5. Requests operational or configuration information, or changes configuration information, as described in [“Requesting Operational Information Using NETCONF” on page 175](#), [“Requesting Configuration Data Using NETCONF” on page 184](#), and [“Editing the Candidate Configuration Using NETCONF” on page 83](#).
6. (Optional) Verifies the syntactic correctness of a configuration before attempting to commit it, as described in [“Verifying the Configuration Syntax Using NETCONF” on page 113](#).
7. Commits changes made to the configuration, as described in [“Committing the Candidate Configuration Using NETCONF” on page 114](#) and [“Committing the Candidate Configuration Only After Confirmation Using NETCONF” on page 114](#).
8. Unlocks the candidate configuration if it is locked, as described in [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#).
9. Ends the NETCONF session and closes the connection to the device, as described in [“Ending a NETCONF Session and Closing the Connection” on page 75](#).

Generating Well-Formed XML Documents

Each set of NETCONF and Junos XML tag elements emitted by the NETCONF server and a client application within a `<hello>`, `<rpc>`, or `<rpc-reply>` tag element must constitute a well-formed XML document by obeying the structural rules defined in the document type definition (DTD) for the kind of information being sent. The client application must emit tag elements in the required order and only in the allowed contexts.

The NETCONF server and client applications must also comply with RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, available at <http://www.ietf.org/rfc/rfc4742.txt>. In particular, the server and applications must send the character sequence `]]>]]>` after each XML document. Because this sequence is not legal within an XML document, it unambiguously signals the end of a document. In practice, the client application sends the sequence after the closing `</hello>` tag and each closing `</rpc>` tag, and the NETCONF server sends it after the closing `</hello>` tag and each closing `</rpc-reply>` tag.



NOTE: In the following example (and in all examples in this document of tag elements emitted by a client application), bold font is used to highlight the part of the tag sequence that is discussed in the text.

```
<!-- generated by a client application -->
<hello | rpc>
  <!-- contents of top-level tag element -->
</hello | /rpc>
]]>]]>

<!-- generated by the NETCONF server -->
<hello | rpc-reply attributes>
  <!-- contents of top-level tag element -->
</hello | /rpc-reply>
]]>]]>
```

- Related Documentation**
- [Connecting to the NETCONF Server on page 60](#)
 - [Starting the NETCONF Session on page 61](#)

Understanding the Request Procedure in a NETCONF Session

You can use the NETCONF XML management protocol and Junos XML API to request information about the status and the current configuration of a routing, switching, or security platform running Junos OS. The tags for operational requests are defined in the Junos XML API and correspond to Junos OS command-line interface (CLI) operational commands. There is a request tag element for many commands in the CLI **show** family of commands.

The tag element for configuration requests is the NETCONF **<get-config>** tag element. It corresponds to the CLI configuration mode **show** command. The Junos XML tag elements that make up the content of both the client application's requests and the NETCONF server's responses correspond to CLI configuration statements, which are described in the Junos OS configuration guides.

In addition to information about the current configuration, client applications can request other configuration-related information, including information about previously committed (rollback) configurations, information about the rescue configuration or an XML schema representation of the configuration hierarchy.

To request information from the NETCONF server, a client application performs the procedures described in the indicated sections:

1. Establishes a connection to the NETCONF server on the routing, switching, or security platform, as described in [“Connecting to the NETCONF Server” on page 60](#).
2. Opens a NETCONF session, as described in [“Starting the NETCONF Session” on page 61](#).
3. If making configuration requests, optionally locks the candidate configuration, as described in [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#).
4. Makes any number of requests one at a time, freely intermingling operational and configuration requests. See [“Requesting Operational Information Using NETCONF” on page 175](#) and [“Requesting Configuration Data Using NETCONF” on page 184](#). The application can also intermix requests with configuration changes.
5. Accepts the tag stream emitted by the NETCONF server in response to each request and extracts its content, as described in [“Parsing the NETCONF Server Response” on page 67](#).
6. Unlocks the candidate configuration if it is locked, as described in [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#). Other users and applications cannot change the configuration while it remains locked.
7. Ends the NETCONF session and closes the connection to the device, as described in [“Ending a NETCONF Session and Closing the Connection” on page 75](#).

CHAPTER 4

NETCONF Perl Client

- [Understanding the NETCONF Perl Distribution and Sample Scripts on page 27](#)

Understanding the NETCONF Perl Distribution and Sample Scripts

Juniper Networks provides a Perl module, called **NET::Netconf::Manager**, to help you more quickly and easily develop custom Perl scripts for configuring and monitoring switches, routers, and security devices running Junos OS . The module implements a **NET::Netconf::Manager** object that client applications can use to communicate with the NETCONF server on a device. The Perl distribution includes several sample Perl scripts, which illustrate how to use the module in scripts that perform various functions.

The NETCONF Perl distribution uses the same directory structure for Perl modules as the [Comprehensive Perl Archive Network](#). This includes a **lib** directory for the **NET::Netconf** module and its supporting files, and an **examples** directory for the sample scripts.

Client applications use the **NET::Netconf::Manager** object to communicate with a NETCONF server. All of the sample scripts use this object.

The sample scripts illustrate how to perform the following functions:

- **diagnose_bgp.pl**—Illustrates how to write scripts to monitor device status and diagnose problems. The sample script extracts and displays information about a device's unestablished Border Gateway Protocol (BGP) peers from the full set of BGP configuration data. The script is in the **examples/diagnose_bgp** directory in the NETCONF Perl distribution.
- **get_chassis_inventory.pl**—Illustrates how to use a predefined query to request information from a device. The sample script invokes the **get_chassis_inventory** query with the **detail** option to request the same information as the Junos XML **get-chassis-inventorydetail/get-chassis-inventory** tag sequence and the command-line interface (CLI) operational mode command **show chassis hardware detail**. The script is in the **examples/get_chassis_inventory** directory in the NETCONF Perl distribution.
- **edit_configuration.pl**—Illustrates how to change the device configuration by loading a file that contains configuration data formatted with Junos XML tag elements. The distribution includes a sample configuration file, **config.xml**; however, you can specify a different configuration file on the command line. The script is in the **examples/edit_configuration** directory in the NETCONF Perl distribution.

For instructions on running the scripts, see the **README** or **README.html** file included in the NETCONF Perl distribution.

**Related
Documentation**

- [Downloading the NETCONF Perl Client and Prerequisites Package on page 35](#)
- [Installing the NETCONF Perl Client and Prerequisites Package on page 36](#)
- [Writing NETCONF Perl Client Applications on page 215](#)

CHAPTER 5

YANG

- [Understanding YANG on Devices Running Junos OS on page 29](#)
- [YANG Modules Overview on page 30](#)

Understanding YANG on Devices Running Junos OS

The NETCONF configuration protocol is an XML-based protocol that client applications use to request information from and make configuration changes to routing, switching, and security devices. YANG is a standards-based, extensible, hierarchical data modeling language that is used to model the configuration and state data used by NETCONF operations, remote procedure calls (RPCs), and server event notifications. The NETMOD working group in the IETF specifically designed YANG to model network management data and to provide a standard for the content layer of the NETCONF model. A YANG module defines a single data model and determines the XML encoding for the data used in NETCONF operations.

Juniper Networks provides YANG modules that define the Junos OS configuration hierarchy and Junos OS YANG extensions. You can download the YANG modules from the Juniper Networks website, or you can generate the modules on the device running Junos OS. When downloaded from the website, the YANG module for the Junos OS configuration hierarchy defines the complete hierarchy for all devices running that Junos OS release. When generated on the local device, the YANG module defines the hierarchy specific to that device.

YANG is similar in many respects to other common programming languages. It uses a C-like syntax, a hierarchical organization of data, and provides a set of built-in types as well as the capability to define derived types. YANG stresses readability, and it provides modularity and flexibility through the use of modules and submodules and reusable types and node groups.

A YANG module defines a data model through its data, and the hierarchical organization of and constraints on that data. A module can be a complete, standalone entity, or it can reference definitions in other modules and submodules as well as augment other data models with additional nodes. The module dictates how the data is represented in XML.

A YANG module defines not only the syntax but also the semantics of the data. It explicitly defines relationships between and constraints on the data. This enables you to create syntactically correct configuration data that meets constraint requirements and enables

you to validate the data against the model before uploading it and committing it on a device.

YANG uses modules to define configuration and state data, notifications, and RPCs for NETCONF in a manner similar to how the Structure of Management Information (SMI) uses MIBs to model data for SNMP operations. However, YANG has the benefit of being able to distinguish between operational and configuration data. YANG maintains compatibility with SNMP's SMI version 2 (SMIv2), and you can use libsmi to translate SMIv2 MIB modules into YANG modules and vice versa. Additionally, when you cannot use a YANG parser, you can translate YANG modules into YANG Independent Notation (YIN), which is an equivalent XML syntax that can be read by XML parsers and XSLT scripts.

You can use existing YANG-based tools or develop custom network management applications to utilize YANG modules for faster and more accurate network programmability. For example, a client application could leverage YANG modules to generate vendor-specific configuration data for different devices and validate that data before uploading it to the device. The application could also handle and troubleshoot unexpected RPC responses and errors.



NOTE: The NETCONF server on devices running Junos OS currently does not support the YANG XML namespace.

For information about NETCONF, see [RFC 6241](#), *NETCONF Configuration Protocol*.

For information about YANG, see [RFC 6020](#), *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, and related RFCs.

**Related
Documentation**

- [YANG Modules Overview on page 30](#)
- [Using Juniper Networks YANG Modules on page 235](#)
- [show system schema on page 238](#)

YANG Modules Overview

YANG data models comprise modules and submodules and can define configuration and state data, notifications, and RPCs for use by NETCONF-based operations. A YANG module defines a data model through its data, and the hierarchical organization of and constraints on that data. Each module is uniquely identified by a namespace URL.

A module defines a single data model. However, a module can reference definitions in other modules and submodules by using the **import** statement to import external modules or the **include** statement to include one or more submodules. Additionally, a module can augment another data model by using the **augment** statement to define the placement of the new nodes in the data model hierarchy and the **when** statement to define the conditions under which the new nodes are valid. A module uses the **feature** statement to specify parts of a module that are conditional and the **deviation** statement to specify where the device's implementation might deviate from the original definition.

When you import an external module, you define a prefix that is used when referencing definitions in the imported module. We recommend that you use the same prefix as that defined in the imported module to avoid conflicts.

YANG models data using a hierarchical, tree-based structure with nodes. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes. The nodes types are:

- leaf node—Contains a single value of a specific type
- leaf-list node—Contains a sequence of leaf nodes
- container node—Contains a grouping of related nodes containing only child nodes, which can be any of the four node types
- list node—Contains a sequence of list entries, each of which is uniquely identified by one or more key leaves

In YANG, each leaf and leaf-list node includes the **type** statement to identify the data type for valid data for that node. YANG defines a set of built-in types and also provides the **typedef** statement for defining a derived type from a base type, which can be either a built-in type or another derived type.

By default, a node defines configuration data. A node defines state data if it is tagged as **config false**. Configuration data is returned using the NETCONF **<get-config>** operation, and state data is returned using the NETCONF **<get>** operation.

**Related
Documentation**

- [Understanding YANG on Devices Running Junos OS on page 29](#)
- [Using Juniper Networks YANG Modules on page 235](#)
- [show system schema on page 238](#)

PART 2

Installation

- [NETCONF Perl Client on page 35](#)
- [NETCONF Java Toolkit on page 41](#)

CHAPTER 6

NETCONF Perl Client

- Downloading the NETCONF Perl Client and Prerequisites Package on page 35
- Installing the NETCONF Perl Client and Prerequisites Package on page 36

Downloading the NETCONF Perl Client and Prerequisites Package

To download the compressed tar archives that contains the NETCONF Perl client distribution and the prerequisites package, perform the following steps:

1. Access the download page for NETCONF at <http://www.juniper.net/support/downloads/?p=netconf>.
2. Select the appropriate software release.
3. Select the Software tab.
4. Click the links labeled NETCONF API Perl client and NETCONF API Perl client prerequisites to download the client distribution and the prerequisites package.



NOTE: The NETCONF XML protocol Perl client software should be installed and run on a regular computer with a UNIX-like operating system; it is not meant to be installed on a Juniper Networks device.

Optionally, download the packages containing the document type definitions (DTDs) and the XML Schema language representation of the Junos OS configuration hierarchy:

1. Access the download page for the Junos XML API at <http://www.juniper.net/support/downloads/?p=junosxml>.
2. Select the appropriate software release.
3. Select the Software tab.
4. Click the links to download the desired packages.

Related Documentation

- Installing the NETCONF Perl Client and Prerequisites Package on page 36
- Writing NETCONF Perl Client Applications on page 215

Installing the NETCONF Perl Client and Prerequisites Package

To install the NETCONF Perl client and the prerequisites package, perform the following procedures:

- [Verifying the Installation and Version of Perl on page 36](#)
- [Extracting the NETCONF Perl Client and Sample Scripts on page 36](#)
- [Extracting and Installing the NETCONF Perl Client Prerequisites Package on page 37](#)
- [Installing the NETCONF Perl Client on page 39](#)

Verifying the Installation and Version of Perl

Perl must be installed on your system before you install the NETCONF Perl client prerequisites package or client software. The NETCONF Perl client requires Perl version 5.6.1 or later. To confirm whether Perl is installed on your system and to determine which version of Perl is currently running, issue the following commands:

```
$ which perl
$ perl -v
```

If the issued output indicates that Perl is not installed or that the version is earlier than the required version, you must download and install Perl version 5.6.1 or later in order to use the NETCONF Perl client. The Perl source packages are located at:

<http://www.cpan.org/src/> .

After installing a suitable version of Perl, extract the NETCONF Perl client, extract and install the prerequisites package, and then install the NETCONF Perl client application.

Extracting the NETCONF Perl Client and Sample Scripts

To uncompress and extract the contents of the compressed tar archive that contains the NETCONF Perl client and sample scripts:

1. Create the directory where you want to store the **NET::Netconf** Perl client application and sample scripts, move the downloaded client application file into that directory, and then make that directory the working directory.

```
$ mkdir parent-directory
$ mv netconf-perl-release.tar.gz parent-directory
$ cd parent-directory
```

2. Uncompress and extract the contents of the NETCONF Perl client package:

- On FreeBSD and Linux systems:

```
$ tar xzf netconf-perl-release.tar.gz
```

- On Solaris systems:

```
$ gzip -dc netconf-perl-release.tar.gz | tar xf
```

where *release* is the release code, for example 10.4R2.6. The command creates a directory called **netconf-perl-release** and extracts the contents of the tar archive to it. For example,

a typical filename for the compressed tar archive is **netconf-perl-10.4R2.6.tar.gz**. Extracting the contents of this archive creates the directory **netconf-perl-10.4R2.6** directly under **parent-directory** and places the application files and sample scripts into this new directory.

The **netconf-perl-release/README** file contains instructions for extracting and installing the Perl prerequisite modules, creating a **Makefile**, and installing and testing the **NET::Netconf** module.

Extracting and Installing the NETCONF Perl Client Prerequisites Package

The prerequisites package consists of C libraries, executables, and Perl modules. It must be installed on the client machine in order for the NETCONF Perl client and the included examples to work correctly. The NETCONF Perl distribution includes the **install-prereqs.pl** script, which you use to install the prerequisites. Starting with Junos OS Release 11.4, you have the option to install all Perl modules that are part of the prerequisites directly from the Comprehensive Perl Archive Network (CPAN) global repository.

To uncompress and extract the contents of the compressed tar archive containing the prerequisite files:

1. Move the downloaded prerequisites package into the **parent-directory/netconf-perl-release/** directory that was created in [“Extracting the NETCONF Perl Client and Sample Scripts”](#) on page 36.

The compressed tar archive containing the prerequisite files must be uncompressed, unpacked, and installed in that directory.

2. Uncompress and extract the contents of the package:

- On FreeBSD and Linux systems:

```
$ tar xzf netconf-perl-prereqs-release.tar.gz
```

- On Solaris systems:

```
$ gzip -dc netconf-perl-prereqs-release.tar.gz | tar xf
```

where *release* is the release code, for example 10.4R2.6. This command creates a directory called **prereqs/** and extracts the contents of the tar archive to it.

By default, the prerequisite Perl modules are installed in the standard directory, which is **/usr/local/lib/**. You can opt to install the modules in a private directory.

- To install the required modules in the standard directory:

1. Log in as root.
2. Go to the **netconf-perl-release/** directory where you extracted the contents of the prerequisites package.
3. Issue the following command:

```
# perl install-prereqs.pl -used_by example -force
```

where the **-used_by example** option is invoked to install only modules used by a specific example, and the **-force** option installs the module even if an earlier version of the module exists or if the **make test** command fails.

- To install the required modules in a private directory:

1. Set the **PERL5LIB**, **MANPATH**, and **PATH** environment variables.

```
$ setenv PERL5LIB private-directory-path
$ setenv MANPATH "$MANPATH/:$PERL5LIB/./man"
$ setenv PATH "$PATH/:$PERL5LIB/./bin"
```

For sh, ksh, and bash shells, **\$PERL5LIB** can be set with **EXPORT**
PERL5LIB=private-directory-path

2. Go to the **netconf-perl-release** directory where you extracted the contents of the prerequisites package.
3. Issue the following command:

```
$ perl install-prereqs.pl -used_by example -install_directory $PERL5LIB -force
```

where the **-used_by example** option is invoked to install only modules used by a specific example, and the **-force** option installs the module even if an earlier version of the module exists or if the **make test** command fails. The **-install_directory \$PERL5LIB** option installs the prerequisite Perl modules in the private directory that you specified in Step 1.

Starting with Junos OS Release 11.4, after issuing the **perl install-prereqs.pl** command, the script provides the option to install the prerequisites from CPAN. The CPAN module is included with standard Perl installations. If you choose to install from CPAN, the script checks that the CPAN module is installed on your system and that you have connectivity to <http://www.cpan.org>. If the CPAN module is present and connectivity is verified, installation begins automatically.

1. To install from CPAN, press Enter or type 'y' when prompted.

```
# perl install-prereqs.pl
This script installs all modules required by default.
Would you like to install the pre-requisite modules from CPAN? [y]/n y
Testing MCPAN on your system...
OK
Trying to ping CPAN
OK
These modules will be installed in the system directory.

This installation takes around 15 minutes

Begin automatic installation:
<output omitted>
```

You might be prompted for additional information during the installation. For example, if additional dependent modules are required for a specific module, the installer might ask if the missing modules should be added to the install queue.



NOTE: On some systems, the firewall might reject utilities that are set to use active FTP, and CPAN installation might hang. If this is an issue, set the corresponding environment variable so that passive FTP is enabled.

Installation log files are written to **netconf-perl-release/tmp/output/**. After installation, you can view any missing dependencies by issuing the following command:

```
$ perl required-mod.pl
```

This command lists the modules that still require installation.

Installing the NETCONF Perl Client

After installing the prerequisites package as detailed in [“Extracting and Installing the NETCONF Perl Client Prerequisites Package” on page 37](#), install the NETCONF Perl client software.

To install the client software:

1. Go to the **netconf-perl-release/** directory that was created in [“Extracting the NETCONF Perl Client and Sample Scripts” on page 36](#).
2. Create the makefile.

- To install the Perl client in the standard directory (usually **/usr/local/lib**):

```
# perl Makefile.PL
```

```
Checking if your kit is complete...
Looks good
Writing Makefile for netconf-perl
```

- To install the Perl client in a private directory:

Make sure that the **PERL5LIB**, **MANPATH**, and **PATH** environment variables are set as detailed in [“Extracting and Installing the NETCONF Perl Client Prerequisites Package” on page 37](#), and create the makefile:

```
# perl Makefile.PL LIB=$PERL5LIB INSTALLMAN3DIR=$PERL5LIB/../man/man3
```

3. Install the **Net::NETCONF** module:

```
# make
# make install
```

The NETCONF Perl client application is installed and ready for use. For information about the **Net::NETCONF::Manager**, **Net::NETCONF::Transform**, or **Net::NETCONF::Trace** classes, consult the appropriate man page by invoking the **man** command and specifying the class. For example:

```
$ man Net::NETCONF::Manager
$ man Net::NETCONF::Transform
$ man Net::NETCONF::Trace
```

The sample scripts reside in the **netconf-perl-release/examples/** directory. You can review and run these examples to acquire some familiarity with the client before writing your own applications.

Related Documentation

- [Downloading the NETCONF Perl Client and Prerequisites Package on page 35](#)
- [Writing NETCONF Perl Client Applications on page 215](#)

CHAPTER 7

NETCONF Java Toolkit

- [Downloading and Installing the NETCONF Java Toolkit on page 41](#)

Downloading and Installing the NETCONF Java Toolkit

A *configuration management server* is a PC or workstation that is used to configure a router, switch, or security device remotely. To use the NETCONF Java toolkit, download and install the toolkit on the configuration management server. The toolkit contains the Netconf.jar library, which is compatible with Java Version 1.4 and later. The following tasks are discussed:

1. [Downloading the NETCONF Java Toolkit on page 41](#)
2. [Installing the NETCONF Java Toolkit on page 41](#)
3. [Satisfying Requirements for SSHv2 Connections on page 41](#)

Downloading the NETCONF Java Toolkit

To download the NETCONF Java toolkit to the configuration management server:

1. Access the GitHub download page at <https://github.com/Juniper/netconf-java/releases>.
2. Download the **Netconf.jar** file.

Installing the NETCONF Java Toolkit

To install the NETCONF Java toolkit on the configuration management server:

1. Include the **Netconf.jar** file in the CLASSPATH of your local Java development environment.
2. Ensure SSHv2/NETCONF connectivity to the device on which the NETCONF server is running.

Satisfying Requirements for SSHv2 Connections

The NETCONF server communicates with client applications within the context of a NETCONF session. The server and client explicitly establish a connection and session before exchanging data, and close the session and connection when they are finished.

The NETCONF Java toolkit accesses the NETCONF server using the SSH protocol and uses the standard SSH authentication mechanism. To establish an SSHv2 connection with a device running Junos OS, you must ensure that the following requirements are met:

- The client application has a user account and can log in to each device where a NETCONF session will be established.
- The login account used by the client application has an SSH public/private key pair or a text-based password.
- The client application can access the public/private keys or text-based password.
- The NETCONF service over SSH is enabled on each device where a NETCONF session will be established.

For information about enabling NETCONF on a device running Junos OS and satisfying the requirements for establishing an SSH session, see the *NETCONF XML Management Protocol Developer Guide*.

For information about NETCONF over SSH, see RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, which is available at <http://www.ietf.org/rfc/rfc4742.txt>.

**Related
Documentation**

- *Creating and Executing a NETCONF Java Application*
- *NETCONF Java Toolkit Overview*
- [NETCONF XML Management Protocol and Junos XML API Overview on page 3](#)

PART 3

Configuration

- [NETCONF Tracing Operations on page 45](#)
- [NETCONF Session on page 51](#)
- [Changing the Configuration on page 83](#)
- [Committing the Configuration on page 113](#)
- [Configuration Statements on page 117](#)
- [NETCONF Protocol Operations on page 127](#)
- [NETCONF Request and Response Tags on page 137](#)
- [Junos XML Protocol Elements Supported in NETCONF Sessions on page 143](#)
- [Junos XML Protocol Element Attributes Supported in NETCONF Sessions on page 167](#)

CHAPTER 8

NETCONF Tracing Operations

- [NETCONF Tracing Operations Overview on page 45](#)
- [Example: Configuring NETCONF Tracing Operations on page 46](#)

NETCONF Tracing Operations Overview

You can configure tracing operations for the NETCONF XML management protocol. NETCONF tracing operations record NETCONF session data in a trace file. By default, NETCONF tracing operations are not enabled.

You configure NETCONF tracing operations at the `[edit system services netconf traceoptions]` hierarchy level.

```
[edit system services]
netconf {
  traceoptions {
    file <filename> <files number> <match regular-expression> <size size>
      <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
    on-demand;
  }
}
```

To enable NETCONF tracing operations and to trace all incoming and outgoing data from NETCONF sessions on that device, configure the **flag all** statement. You can restrict tracing to only incoming or outgoing NETCONF data by configuring the flag value as either **incoming** or **outgoing**, respectively. Additionally, to restrict the trace output to include only those lines that match a particular expression, configure the **file match** statement and define the regular expression against which the output is matched.

NETCONF tracing operations record NETCONF session data in the file `/var/log/netconf`. To specify a different trace file, configure the **file** statement and desired filename.

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 limits on the number and size of trace files by including the **file files number** and **file size size** statements. You can configure up to a maximum of 1000 files. Specify the file size in bytes or use **sizek** to specify KB, **sizem** to specify MB, or **sizeg** to specify GB. You cannot configure the maximum

number of trace files and the maximum trace file size independently. If one option is configured, the other option must also be configured along with a filename.

To control the tracing operation from within a NETCONF session, configure the **on-demand** statement. This requires that you start and stop tracing operations from within the NETCONF session. If you configure the **on-demand** statement, you must issue the `<rpc><request-netconf-trace><start/></request-netconf-trace></rpc>` RPC in the NETCONF session to start tracing operations for that session. To stop tracing for that NETCONF session, issue the `<rpc><request-netconf-trace><stop/></request-netconf-trace></rpc>` RPC.

By default, access to the NETCONF trace file is restricted to the owner. You can manually configure access by including either the **world-readable** or **no-world-readable** statement. The **no-world-readable** statement restricts trace file access to the owner. This is the default. The **world-readable** statement enables unrestricted access to the trace file.

Related Documentation

- [Example: Configuring NETCONF Tracing Operations on page 46](#)
- [netconf on page 119](#)
- [ssh \(NETCONF\) on page 123](#)
- [traceoptions \(NETCONF\) on page 124](#)

Example: Configuring NETCONF Tracing Operations

This example demonstrates how to configure tracing operations for NETCONF sessions.

- [Requirements on page 46](#)
- [Overview on page 46](#)
- [Configuration on page 47](#)
- [Verification on page 48](#)

Requirements

- A routing, switching, or security device running Junos OS Release 12.2 or later is required.

Overview

This example configures basic tracing operations for NETCONF sessions. The example configures the trace file **netconf-ops.log** and sets a maximum number of 20 trace files and a maximum size of 3 MB for each file. The **flag all** statement configures tracing for all incoming and outgoing NETCONF data. The **world-readable** option enables unrestricted access to the NETCONF trace files.

Configuration

CLI Quick Configuration To quickly configure this example, copy the following commands, paste them in a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level.

```
set system services netconf ssh
set system services netconf traceoptions file netconf-ops.log
set system services netconf traceoptions file size 3m
set system services netconf traceoptions file files 20
set system services netconf traceoptions file world-readable
set system services netconf traceoptions flag all
```

Configuring NETCONF Tracing Operations

Step-by-Step Procedure To configure NETCONF tracing operations:

1. Enable NETCONF over SSH.


```
[edit]
user@R1# set system services netconf ssh
```
2. Configure the traceoptions flag to specify which NETCONF session data to capture.

 You can specify incoming, outgoing, or all. This example configures tracing for all NETCONF session data.


```
[edit]
user@R1# set system services netconf traceoptions flag all
```
3. (Optional) Configure the filename of the trace file.

 The following statement configures the trace file **netconf-ops.log**, which is stored in the **/var/log** directory. If you do not specify a filename, NETCONF session data is stored in **/var/log/netconf**.


```
[edit]
user@R1# set system services netconf traceoptions file netconf-ops.log
```
4. (Optional) Configure the maximum number of NETCONF trace files and the maximum size of each file.

 The following statements configure a maximum of 20 trace files with a maximum size of 3 MB per file.


```
[edit]
user@R1# set system services netconf traceoptions file files 20
user@R1# set system services netconf traceoptions file size 3m
```
5. (Optional) Restrict the trace output to include only those lines that match a particular regular expression.

 The following configuration, which is not used in this example, matches on and logs only NETCONF session data that contains "error-message".


```
[edit]
user@R1# set system services netconf traceoptions file match error-message
```

6. (Optional) Configure on-demand tracing to control tracing operations from the NETCONF session.

The following configuration, which is not used in this example, enables on-demand tracing.

```
[edit]
user@R1# set system services netconf traceoptions on-demand
```

7. (Optional) Configure the permissions on the trace file by specifying whether the file is **world-readable** or **no-world-readable**.

This example enables unrestricted access to the trace file.

```
[edit]
user@R1# set system services netconf traceoptions file world-readable
```

8. Commit the configuration.

```
[edit]
user@R1# commit
```

Results

```
[edit]
system {
  services {
    netconf {
      ssh;
      traceoptions {
        file netconf-ops.log size 3m files 20 world-readable;
        flag all;
      }
    }
  }
}
```

Verification

Verifying NETCONF Tracing Operation

Purpose Verify that the device is writing NETCONF session data to the configured trace file. This example logs both incoming and outgoing NETCONF data. In the NETCONF session, which is not detailed here, the user modifies the candidate configuration on R1 to include the **bgp-troubleshoot.slax** op script and then commits the configuration.

Action Display the trace output of the configured NETCONF trace file `/var/log/netconf-ops.log` by issuing the **show log** operational mode command.

```

user@R1 show log netconf-ops.log
Apr  3 13:09:04 Started tracing session: 3694
Apr  3 13:09:29 [3694] Incoming: <rpc>
Apr  3 13:09:29 [3694] Outgoing: <rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/12.2R1/junos">
Apr  3 13:09:39 [3694] Incoming: <edit-config>
Apr  3 13:09:43 [3694] Incoming: <target>
Apr  3 13:09:47 [3694] Incoming: <candidate/>
Apr  3 13:09:53 [3694] Incoming: </target>
Apr  3 13:10:07 [3694] Incoming: <default-operation>merge</default-operation>
Apr  3 13:10:10 [3694] Incoming: <config>
Apr  3 13:10:13 [3694] Incoming: <configuration>
Apr  3 13:10:16 [3694] Incoming: <system>
Apr  3 13:10:19 [3694] Incoming: <scripts>
Apr  3 13:10:23 [3694] Incoming: <op>
Apr  3 13:10:26 [3694] Incoming: <file>
Apr  3 13:10:44 [3694] Incoming: <name>bgp-troubleshoot.slax</name>
Apr  3 13:10:46 [3694] Incoming: </file>
Apr  3 13:10:48 [3694] Incoming: </op>
Apr  3 13:10:52 [3694] Incoming: </scripts>
Apr  3 13:10:56 [3694] Incoming: </system>
Apr  3 13:11:00 [3694] Incoming: </configuration>
Apr  3 13:11:00 [3694] Outgoing: <ok/>
Apr  3 13:11:12 [3694] Incoming: </config>
Apr  3 13:11:18 [3694] Incoming: </edit-config>
Apr  3 13:11:26 [3694] Incoming: </rpc>
Apr  3 13:11:26 [3694] Outgoing: </rpc-reply>
Apr  3 13:11:26 [3694] Outgoing: ]]>]]>
Apr  3 13:11:31 [3694] Incoming: ]]>]]>

Apr  3 13:14:20 [3694] Incoming: <rpc>
Apr  3 13:14:20 [3694] Outgoing: <rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/12.2R1/junos">
Apr  3 13:14:26 [3694] Incoming: <commit/>
Apr  3 13:14:35 [3694] Outgoing: <ok/>
Apr  3 13:14:35 [3694] Incoming: </rpc>
Apr  3 13:14:35 [3694] Outgoing: </rpc-reply>
Apr  3 13:14:35 [3694] Outgoing: ]]>]]>
Apr  3 13:14:40 [3694] Incoming: ]]>]]>

Apr  3 13:30:48 [3694] Outgoing: <!-- session end at 2012-04-03 13:30:48 PDT -->

```

Meaning This example configured the **flag all** statement, so the trace file displays all incoming and outgoing NETCONF session operations. Each operation includes the date and timestamp. Multiple NETCONF sessions are distinguished by a session number. In this example, only one session, 3694, is active.

Related Documentation

- [NETCONF Tracing Operations Overview on page 45](#)
- [netconf on page 119](#)
- [ssh \(NETCONF\) on page 123](#)

- [traceoptions \(NETCONF\) on page 124](#)

CHAPTER 9

NETCONF Session

- [Establishing an SSH Connection for a NETCONF Session on page 51](#)
- [Connecting to the NETCONF Server on page 60](#)
- [Starting the NETCONF Session on page 61](#)
- [Sending a Request to the NETCONF Server on page 65](#)
- [Parsing the NETCONF Server Response on page 67](#)
- [Using a Standard API to Parse Response Tag Elements in NETCONF and Junos XML Protocol Sessions on page 70](#)
- [Handling an Error or Warning in a NETCONF Session on page 70](#)
- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)
- [Terminating a NETCONF Session on page 74](#)
- [Ending a NETCONF Session and Closing the Connection on page 75](#)
- [Sample NETCONF Session on page 76](#)
- [Configuring RFC-Compliant NETCONF Sessions on page 79](#)

Establishing an SSH Connection for a NETCONF Session

- [Establishing an SSH Connection for a NETCONF Session on page 51](#)
- [Prerequisites for Establishing an SSH Connection for NETCONF Sessions on page 52](#)
- [Prerequisites for Establishing an Outbound SSH Connection for NETCONF Sessions on page 56](#)

Establishing an SSH Connection for a NETCONF Session

You use the SSH protocol to establish connections between a *configuration management server* and a device running Junos OS. A configuration management server, as the name implies, is used to configure the device running Junos OS remotely. This server typically manages the configurations using Perl scripts.

There are two options available when establishing a connection between the configuration management server and a device running Junos OS: SSH and outbound SSH. With SSH, the configuration management server initiates an SSH session with the device running Junos OS. Outbound SSH is used when the configuration management server cannot initiate an SSH connection because of network restrictions (such as a firewall). In this

situation, the device running Junos OS is configured to initiate, establish, and maintain an SSH connection with a predefined set of configuration management servers. For a complete discussion of outbound SSH, see *Configuring Outbound SSH Service* in the *Junos OS Administration Library for Routing Devices*.

Prerequisites for Establishing an SSH Connection for NETCONF Sessions

Before the configuration management server establishes an SSH connection with a device running Junos OS, you must satisfy the requirements discussed in the following sections.

1. [Installing SSH Software on the Configuration Management Server on page 52](#)
2. [Configuring a User Account for the Client Application on Devices Running Junos OS on page 52](#)
3. [Configuring a Public/Private Key Pair or Password for the Junos OS User Account on page 53](#)
4. [Accessing the Keys or Password with the Client Application on page 54](#)
5. [Enabling NETCONF Service over SSH on page 55](#)

Installing SSH Software on the Configuration Management Server

The configuration management server handles the SSH connection between the configuration management server and the device running Junos OS. Therefore, the SSH software must be installed locally on the configuration management server.

If the client application accessing the NETCONF server uses the NETCONF Java Toolkit or the NETCONF Perl module provided by Juniper Networks, no further action is necessary. The NETCONF Java Toolkit includes SSH software, and as part of the installation procedure for the NETCONF Perl module, you install a prerequisites package that includes the necessary SSH software. If the client application does not use the NETCONF Java Toolkit or the NETCONF Perl module, obtain the SSH software and install it on the configuration management server where the client application runs. For information about obtaining and installing SSH software, see <http://www.ssh.com/> and <http://www.openssh.com/>.

Configuring a User Account for the Client Application on Devices Running Junos OS

When establishing a NETCONF session, the configuration management server must log in to the device running Junos OS. Thus, each configuration management server needs a user account on each device where a NETCONF session will be established. The following instructions explain how to create a login account on devices running Junos OS. Alternatively, you can skip this section and enable authentication through RADIUS or TACACS+; for instructions, see the *Junos OS Administration Library for Routing Devices*.

To determine whether a login account exists on a device running Junos OS, enter CLI configuration mode on the device and issue the following commands:

```
[edit system login]
user@host# show user account-name
```


If the appropriate account does not exist, perform the following steps to create one:

1. Configure the **user** statement at the **[edit system login]** hierarchy level and specify a username. Include the **class** statement, and specify a login class that has the permissions required for all actions to be performed by the application.

```
[edit system login]
user@host# set user username class class-name
```

2. Optionally, include the **full-name** and **uid** statements at the **[edit system login user *username*]** hierarchy level.
3. Commit the configuration to activate the user account on the device.

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

4. Repeat the preceding steps on each device running Junos OS where the client application establishes NETCONF sessions.

Configuring a Public/Private Key Pair or Password for the Junos OS User Account

The configuration management server needs an SSH public/private key pair, a text-based password, or both before it can authenticate with the NETCONF server. A public/private key pair is sufficient if the account is used only to connect to the NETCONF server through SSH. If the account is also used to access the device in other ways (for login on the console, for example), it must have a text-based password. The password is also used (the SSH server prompts for it) if key-based authentication is configured but fails.



NOTE: You can skip this section if you have chosen to enable authentication through RADIUS or TACACS+, as described in the *Junos OS Administration Library for Routing Devices*.

To create a text-based password, perform the following steps:

1. Include either the **plain-text-password** or **encrypted-password** statement at the **[edit system login user *username* authentication]** hierarchy level.

To enter a password as text, issue the following command. You are prompted for the password, which is encrypted before being stored.

```
[edit system login user username authentication]
user@host# set plain-text-password
New password: password
Retype new password: password
```

To store a password that you have previously created and hashed using Message Digest 5 (MD5) or Secure Hash Algorithm 1 (SHA-1), issue the following command:

```
[edit system login user username authentication]
user@host# set encrypted-password "password"
```

2. Commit the configuration.

```
[edit system login user username authentication]
```

```
user@host# commit
```

3. Repeat the preceding steps on each device running Junos OS where the client application establishes NETCONF sessions.

To create an SSH public/private key pair, perform the following steps:

1. Issue the **ssh-keygen** command in the standard command shell (not the Junos OS CLI) on the configuration management server where the client application runs.

By providing the appropriate arguments, you encode the public key with either RSA (supported by SSH versions 1 and 2) or the Digital Signature Algorithm (DSA, supported by SSH version 2). For more information, see the manual page for the **ssh-keygen** command. Junos OS uses SSH version 2 by default, but also supports version 1.

```
% ssh-keygen options
```

2. Associate the public key with the Junos OS login account by including the **load-key-file** statement at the **[edit system login user account-name authentication]** hierarchy level.

```
[edit system login user username authentication]
user@host# set load-key-file URL
```

Junos OS copies the contents of the specified file onto the device running Junos OS. *URL* is the path to the file that contains one or more public keys. The **ssh-keygen** command by default stores each public key in a file in the **.ssh** subdirectory of the user home directory; the filename depends on the encoding (DSA or RSA) and SSH version. For information about specifying URLs, see the *CLI User Guide*.

Alternatively, you can include one or both of the **ssh-dsa** and **ssh-rsa** **ssh-ds** statements at the **[edit system login user account-name authentication]** hierarchy level. We recommend using the **load-key-file** statement, however, because it eliminates the need to type or cut-and-paste the public key on the command line. For more information about the **ssh-dsa** and **ssh-rsa** statements, see the *Junos OS Administration Library for Routing Devices*.

3. Commit the configuration.

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

4. Repeat Step 2 and Step 3 on each device running Junos OS where the client application establishes NETCONF sessions.

Accessing the Keys or Password with the Client Application

The client application must be able to access the public/private keys or password you created in [“Configuring a Public/Private Key Pair or Password for the Junos OS User Account” on page 53](#) and provide it when the NETCONF server prompts for it.

There are several methods for enabling the application to access the key or password:

- If public/private keys are used, the ssh-agent program runs on the computer where the client application runs, and handles the private key.
- When a user starts the application, the application prompts the user for the password and stores it temporarily in a secure manner.
- The password is stored in encrypted form in a secure local-disk location or in a secured database.

Enabling NETCONF Service over SSH

RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, requires that the NETCONF server, by default, provide the client device with access to the NETCONF SSH subsystem when the SSH session is established over a dedicated IANA-assigned TCP port. Use of a dedicated port makes it easy to identify and filter NETCONF traffic. The IANA-assigned port for NETCONF-over-SSH sessions is 830.

You also can configure the server to allow access to the NETCONF SSH subsystem either over the default SSH port (22) or over a port number that is explicitly configured. An explicitly configured port accepts only NETCONF-over-SSH sessions and rejects regular SSH session requests. If SSH services are enabled on the server, the default SSH port (22) continues to accept NETCONF sessions even when an alternate NETCONF-over-SSH port is configured. For added security, you can configure event policies that utilize **UI_LOGIN_EVENT** information to effectively disable the default port or further restrict NETCONF server access on a port.

To enable NETCONF service over SSH on a device running Junos OS, perform the following steps:

1. Include one of the following statements at the indicated configuration hierarchy level:
 - To enable access to the NETCONF SSH subsystem using the default NETCONF-over-SSH port (830) as specified by RFC 4742, include the **ssh** statement at the **[edit system services netconf]** hierarchy level:

```
[edit system services]
user@host# set netconf ssh
```

- To enable access to the NETCONF SSH subsystem using a specified port number, configure the **port** statement with the desired port number at the **[edit system services netconf ssh]** hierarchy level.

```
[edit system services]
user@host# set netconf ssh port port-number
```

The ***port-number*** can range from 1 through 65535. The configured port accepts only NETCONF-over-SSH sessions and rejects regular SSH session requests.



NOTE: Although NETCONF-over-SSH can be configured on any port from 1 through 65535, you should avoid configuring access on a port that is normally assigned for another service. This practice avoids potential resource conflicts. If you configure NETCONF-over-SSH on a port assigned for another service, such as FTP, and that service is enabled, a commit check does not reveal a resource conflict or issue any warning message to that effect.

- To enable access to the NETCONF SSH subsystem using the default SSH port (22), include the **ssh** statement at the **[edit system services]** hierarchy level. This configuration enables SSH access to the device for all users and applications. The **ssh** statement can be included in the configuration in addition to the configuration statements listed previously.

```
[edit system services]
user@host# set ssh
```

2. Commit the configuration:

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

3. Repeat the preceding steps on each device running Junos OS where the client application establishes NETCONF sessions.

Prerequisites for Establishing an Outbound SSH Connection for NETCONF Sessions

To enable a configuration management server to establish an outbound SSH connection to the NETCONF server, you must satisfy the requirements discussed in the following sections:

1. [Configuring the Device Running Junos OS for Outbound SSH on page 56](#)
2. [Installing SSH Software on the Client on page 58](#)
3. [Receiving and Managing the Outbound SSH Initiation Sequence on the Client on page 59](#)
4. [Enabling NETCONF Service over SSH on page 59](#)

Configuring the Device Running Junos OS for Outbound SSH

To configure the device running Junos OS for outbound SSH:

1. At the **[edit system services ssh]** hierarchy level, set the SSH **protocol-version** to v2:

```
[edit system services ssh]
user@host# set protocol-version v2
```

2. Generate or obtain a public/private key pair for the device running Junos OS. This key pair will be used to encrypt the data transferred across the SSH connection. For more information on generating key pairs, see the *Junos OS Administration Library for Routing Devices*.

3. If you are manually installing the public key on the configuration management server, transfer the public key to the configuration management server.
4. At the **[edit system services]** hierarchy level, include the **outbound-ssh** configuration hierarchy and any required statements.

```
[edit system services]
outbound-ssh {
  client client-id {
    address {
      port port-number;
      retry number;
      timeout seconds;
    }
    device-id device-id;
    keep-alive {
      retry number
      timeout seconds;
    }
    reconnect-strategy (sticky | in-order);
    secret password;
    services netconf;
  }
}
```

The attributes are as follows:

address—(Required) The hostname or IPv4 address or IPv6 address of the configuration management server. You can list multiple clients by adding each client's hostname or IP address along with the connection parameters listed below:

- **port *port-number***—Port at which a server listens for outbound SSH connection requests. The default is port 22.
- **retry *number***—Maximum number of connection attempts the device running Junos OS makes to the specified IP address to establish an outbound SSH connection. The default is 3.
- **timeout *seconds***—Amount of time, in seconds, that the device running Junos OS attempts to establish an outbound SSH connection. The default is 15 seconds per attempt.



NOTE: Starting with Release 15.1, Junos OS supports outbound-SSH connections with devices having IPv6 addresses.

client client-id—**outbound-ssh** configuration stanza on the device. Each **outbound-ssh** stanza represents a single outbound SSH connection. The *client-id* value is not sent to the client.

device-id device-id—Unique ID identifying the device running Junos OS to the configuration management server during the initiation process.

keep-alive—(Optional) Specify that keepalive messages be sent from the device running Junos OS to the configuration management server. To configure the keepalive message, you must configure both the **timeout** and **retry** statements.

- **retry number**—Number of keepalive messages the device running Junos OS sends without receiving a response from the configuration management server before the current SSH connection is terminated. The default is three tries.
- **timeout seconds**—Amount of time, in seconds, that the server waits for data before sending a keepalive signal. The default is 15 seconds.

reconnect-strategy (sticky | in-order)—(Optional) Method that the device running Junos OS uses to reestablish a disconnected outbound SSH connection. Two methods are available:

- **sticky**—Configures the device to reconnect to the configuration management server to which it was last connected. If the server is unavailable, the device attempts to establish a connection with the next configuration management server on the list and so forth until a connection is established.
- **in-order**—Configures the device to reestablish an outbound SSH session based on the configuration management server address list. The device attempts to establish a session with the first server on the list. If this server is unavailable, the device attempts to establish a session with the next configured server. This process repeats until a connection is established.

secret *password*—(Optional) Public SSH host key of the device. If you configure this statement, the device passes its public key to the configuration management server during the initialization of the outbound SSH service. This is the recommended method of maintaining a current copy of the device's public key on the configuration management server.

services netconf—(Required) Configures the application to accept NETCONF as an available service.

5. Commit the configuration:

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

Installing SSH Software on the Client

Once the device establishes the SSH connection to the configuration management server, the configuration management server takes control of the SSH session. Therefore, the SSH client software must be installed locally on the configuration management server.

If the client application accessing the NETCONF server uses the NETCONF Java Toolkit or the NETCONF Perl module provided by Juniper Networks, no further action is necessary. The NETCONF Java Toolkit includes SSH software, and as part of the installation procedure for the NETCONF Perl module, you install a prerequisites package that includes the necessary SSH software. If the client application does not use the NETCONF Java Toolkit or the NETCONF Perl module, obtain the SSH client software and install it on the configuration management server where the application runs. For information about

obtaining and installing SSH software, see <http://www.ssh.com/> and <http://www.openssh.com/>.

Receiving and Managing the Outbound SSH Initiation Sequence on the Client

When configured for outbound SSH, the device running Junos OS attempts to maintain a constant connection with a configuration management server. Whenever an outbound SSH session is not established, the device sends an outbound SSH initiation sequence to a configuration management server listed in the device's configuration management server list. Prior to establishing a connection with the device, each configuration management server must be set up to receive this initiation sequence, establish a TCP connection with the device, and transmit the device identity back to the device.

The initiation sequence takes one of two forms, depending on how you chose to handle the Junos OS server's public key.

If the public key is installed manually on the configuration management server, the initiation sequence takes the following form:

```
MSG-ID: DEVICE-CONN-INFO\r\n
MSG-VER: V1\r\n
DEVICE-ID: <device-id>\r\n
```

If the public key is forwarded to the configuration management server by the device during the initialization sequence, the sequence takes the following form:

```
MSG-ID: DEVICE-CONN-INFO\r\n
MSG-VER: V1\r\n
DEVICE-ID: : <device-id>\r\n
HOST-KEY: <pub-host-key>\r\n
HMAC: <HMAC(pub-SSH-host-key,<secret>)>\r\n
```

Enabling NETCONF Service over SSH

RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, requires that the NETCONF server, by default, provide the client device with access to the NETCONF SSH subsystem when the SSH session is established over a dedicated IANA-assigned TCP port. Use of a dedicated port makes it easy to identify and filter NETCONF traffic. The IANA-assigned port for NETCONF-over-SSH sessions is 830.

You also can configure the server to allow access to the NETCONF SSH subsystem either over the default SSH port (22) or over a port number that is explicitly configured. An explicitly configured port accepts only NETCONF-over-SSH sessions and rejects regular SSH session requests. If SSH services are enabled on the server, the default SSH port (22) continues to accept NETCONF sessions even when an alternate NETCONF-over-SSH port is configured. For added security, you can configure event policies that utilize **UI_LOGIN_EVENT** information to effectively disable the default port or further restrict NETCONF server access on a port.

To enable NETCONF service over SSH on a device running Junos OS, perform the following steps:

1. Include one of the following statements at the indicated hierarchy level:

- To enable access to the NETCONF SSH subsystem using the default NETCONF-over-SSH port (830) as specified by RFC 4742, include the **ssh** statement at the **[edit system services netconf]** hierarchy level:

```
[edit system services]
user@host# set netconf ssh
```

- To enable access to the NETCONF SSH subsystem using a specified port number, configure the **port** statement with the desired port number at the **[edit system services netconf ssh]** hierarchy level.

```
[edit system services]
user@host# set netconf ssh port port-number
```

The *port-number* can range from 1 through 65535. The configured port accepts only NETCONF-over-SSH sessions and rejects regular SSH session requests.



NOTE: Although NETCONF-over-SSH can be configured on any port from 1 through 65535, you should avoid configuring access on a port that is normally assigned for another service. This practice avoids potential resource conflicts. If you configure NETCONF-over-SSH on a port assigned for another service, such as FTP, and that service is enabled, a commit check does not reveal a resource conflict or issue any warning message to that effect.

- To enable access to the NETCONF SSH subsystem using the default SSH port (22), include the **ssh** statement at the **[edit system services]** hierarchy level. This configuration enables SSH access to the device for all users and applications. The **ssh** statement can be included in the configuration in addition to the configuration statements listed previously.

```
[edit system services]
user@host# set ssh
```

2. Commit the configuration:

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

3. Repeat the preceding steps on each device running Junos OS where the client application establishes a NETCONF session.

Connecting to the NETCONF Server

Before a client application can connect to the NETCONF server, you must satisfy the requirements described in [“Establishing an SSH Connection for a NETCONF Session” on page 51](#).

When the prerequisites are satisfied, applications written in Perl use the NETCONF Perl module to connect to the NETCONF server. A client application that does not use the NETCONF Perl module uses one of two methods:

- It uses SSH library routines to establish an SSH connection to the NETCONF server, provide the username and password or passphrase, and create a channel that acts as an SSH subsystem for the NETCONF session. Providing instructions for using library routines is beyond the scope of this document.
- It issues the following **ssh** command to create a NETCONF session as an SSH subsystem:

```
ssh user@hostname -p 830 -s netconf
```

The **-p** option defines the port number on which the NETCONF server listens. This option can be omitted if you enabled access to SSH over the default port in [“Prerequisites for Establishing an SSH Connection for NETCONF Sessions” on page 52](#).

The **-s** option establishes the NETCONF session as an SSH subsystem.

The application must include code to intercept the NETCONF server’s prompt for the password or passphrase. Perhaps the most straightforward method is for the application to use a utility such as the **expect** command. The NETCONF Perl client uses this method, for example.

- Related Documentation**
- [Generating Well-Formed XML Documents on page 24](#)
 - [Starting the NETCONF Session on page 61](#)

Starting the NETCONF Session

Each NETCONF session begins with a handshake in which the NETCONF server and the client application specify the NETCONF capabilities they support. The following sections describe how to start a NETCONF session:

- [Exchanging <hello> Tag Elements on page 61](#)
- [Verifying Compatibility on page 63](#)

Exchanging <hello> Tag Elements

The NETCONF server and client application each begin by emitting a **<hello>** tag element to specify which operations, or *capabilities*, they support from among those defined in the NETCONF specification. The **<hello>** tag element encloses the **<capabilities>** tag element and the **<session-id>** tag element, which specifies the UNIX process ID (PID) of the NETCONF server for the session. Within the **<capabilities>** tag element, a **<capability>** element specifies each supported function.

The client application must emit the **<hello>** tag element before any other tag element during the NETCONF session, and must not emit it more than once.

Each capability defined in the NETCONF specification is represented in a **<capability>** tag element by a uniform resource name (URN). Capabilities defined by individual vendors are represented by uniform resource identifiers (URIs), which can be URNs or URLs. The

NETCONF XML management protocol emits a **<hello>** tag element similar to the following sample output (each **<capability>** tag element appears on three lines for legibility only):

```
<hello>
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>

  <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>

    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>

  <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file</capability>

    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>

  <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>

  <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0</capability>

    <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>

  <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file</capability>

    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>22062</session-id>
</hello>
```

(For information about the **]]>]]>** character sequence, see [“Generating Well-Formed XML Documents” on page 24.](#))

The URIs in the **<hello>** tag element indicate the following supported capabilities:

- **urn:ietf:params:netconf:base:1.0**—The NETCONF server supports the basic operations and tag elements defined in the base NETCONF specification.
- **urn:ietf:params:netconf:capability:candidate:1.0**—The NETCONF server supports operations on a candidate configuration. For more information, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#), [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#), [“Editing the Candidate Configuration Using NETCONF” on page 83](#), [“Rolling Back the Candidate Configuration Using NETCONF” on page 95](#), [“Verifying the Configuration Syntax Using NETCONF” on page 113](#), and [“Committing the Candidate Configuration Using NETCONF” on page 114](#).
- **urn:ietf:params:netconf:capability:confirmed-commit:1.0**—The NETCONF server supports confirmed commit operations. For more information, see [“Committing the Candidate Configuration Only After Confirmation Using NETCONF” on page 114](#).
- **urn:ietf:params:netconf:capability:validate:1.0**—The NETCONF server supports the validation operation, which verifies the syntactic correctness of a configuration without actually committing it. For more information, see [“Verifying the Configuration Syntax Using NETCONF” on page 113](#).

- **urn:ietf:params:netconf:capability:url:1.0?protocol=http,ftp,file**—The NETCONF server accepts configuration data stored in a file. It can retrieve files both from its local filesystem (indicated by the **file** option in the URN) and from remote machines by using Hypertext Transfer Protocol (HTTP) or FTP (indicated by the **http** and **ftp** options in the URN). For more information, see [“Uploading and Formatting Configuration Data in a NETCONF Session” on page 85](#).
- **http://xml.juniper.net/netconf/junos/1.0**—The NETCONF server supports the operations defined in the Junos XML API for requesting and changing operational information (the tag elements in the *Junos XML API Operational Developer Reference*). The NETCONF server also supports operations in the Junos XML management protocol for requesting or changing configuration information.

NETCONF client applications should use only native NETCONF XML management protocol operations and supported extensions available in the Junos XML management protocol for configuration functions. The semantics of corresponding Junos XML protocol operations and NETCONF XML protocol operations are not necessarily identical, so using Junos XML protocol configuration operations other than the documented supported extensions can lead to unexpected results.

- **http://xml.juniper.net/dmi/system/1.0**—The NETCONF server supports the operations defined in the Device Management Interface (DMI) specification.

To comply with the NETCONF specification, the client application also emits a **<hello>** tag element to define the capabilities it supports. It does not include the **<session-id>** tag element:

```
<hello>
<capabilities>
  <capability>first-capability</capability>
  <!-- tag elements for additional capabilities -->
</capabilities>
</hello>
]]>]]>
```

The session continues when the client application sends a request to the NETCONF server. The NETCONF server does not emit any tag elements after session initialization except in response to the client application's requests.

Verifying Compatibility

Exchanging **<hello>** tag elements enables a client application and the NETCONF server to determine if they support the same capabilities. In addition, we recommend that the client application determine the version of the Junos OS running on the NETCONF server. After emitting its **<hello>** tag, the client application emits the **<get-software-information>** tag element in an **<rpc>** tag element:

```
<rpc>
  <get-software-information/>
</rpc>
]]>]]>
```

The NETCONF server returns the **<software-information>** tag element, which encloses the **<host-name>** and **<product-name>** tag elements plus a **<package-information>** tag

element for each Junos OS module. The **<comment>** tag element within the **<package-information>** tag element specifies the Junos OS Release number (in the following example, 8.2 for Junos OS Release 8.2) and the build date in the format YYYYMMDD (year, month, day—12 January 2007 in the following example). Some tag elements appear on multiple lines, for legibility only:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" \
  xmlns:junos="http://xml.juniper.net/junos/8.2R1/junos">
  <software-information>
    <host-name>router1</host-name>
    <product-name>m20</product-name>
    <package-information>
      <name>junos</name>
      <comment>JUNOS Base OS boot [8.2-20070112.0]</comment>
    </package-information>
    <package-information>
      <name>jbase</name>
      <comment>JUNOS Base OS Software Suite \
        [8.2-20070112.0]</comment>
    </package-information>
    <!-- <package-information> tag elements for additional modules -->
  </software-information>
</capabilities>
</rpc-reply>
]]>]]>
```

Normally, the version is the same for all Junos OS modules running on the device (we recommend this configuration for predictable routing performance). Therefore, verifying the version number of just one module is usually sufficient.

The client application is responsible for determining how to handle any differences in version or capabilities. For fully automated performance, include code in the client application that determines whether it supports the same capabilities and Junos OS version as the NETCONF server. Decide which of the following options is appropriate when there are differences, and implement the corresponding response:

- Ignore differences in capabilities and Junos version, and do not alter the client application's behavior to accommodate the NETCONF server. A difference in Junos versions does not necessarily make the server and client incompatible, so this is often a valid approach. Similarly, it is a valid approach if the capabilities that the client application does not support are operations that are always initiated by a client, such as validation of a configuration and confirmed commit. In that case, the client maintains compatibility by not initiating the operation.
- Alter standard behavior to be compatible with the NETCONF server. If the client application is running a later version of the Junos OS, for example, it can choose to emit only NETCONF and Junos XML tag elements that represent the software features available in the NETCONF server's version of the Junos OS.
- End the NETCONF session and terminate the connection. This is appropriate if you decide that it is not practical to accommodate the NETCONF server's version or capabilities. For instructions, see ["Ending a NETCONF Session and Closing the Connection" on page 75](#).

- Related Documentation**
- [Generating Well-Formed XML Documents on page 24](#)
 - [Connecting to the NETCONF Server on page 60](#)
 - [Sending a Request to the NETCONF Server on page 65](#)
 - [Parsing the NETCONF Server Response on page 67](#)

Sending a Request to the NETCONF Server

To initiate a request to the NETCONF server, a client application emits the opening `<rpc>` tag, followed by one or more tag elements that represent the particular request, and the closing `</rpc>` tag, in that order:

```
<rpc>
  <!-- tag elements representing a request -->
</rpc>
]]>]]>
```

The application encloses each request in its own separate pair of opening `<rpc>` and closing `</rpc>` tags. Each request must constitute a well-formed XML document by including only compliant and correctly ordered tag elements. The NETCONF server ignores any newline characters, spaces, or other white space characters that occur between tag elements in the tag stream, but it preserves white space within tag elements.

Optionally, a client application can include one or more attributes of the form ***attribute-name="value"*** in the opening `<rpc>` tag for each request. The NETCONF server echoes each attribute, unchanged, in the opening `<rpc-reply>` tag in which it encloses its response.

A client application can use this feature to associate requests and responses by including an attribute in each opening `<rpc>` request tag that assigns a unique identifier. The NETCONF server echoes the attribute in its opening `<rpc-reply>` tag, making it easy to map the response to the initiating request. The NETCONF specification specifies the name ***message-id*** for this attribute.

Although operational and configuration requests conceptually belong to separate classes, a NETCONF session does not have distinct modes that correspond to CLI operational and configuration modes. Each request tag element is enclosed within its own `<rpc>` tag, so a client application can freely alternate operational and configuration requests. A client application can make three classes of requests:

- [Operational Requests on page 65](#)
- [Configuration Information Requests on page 66](#)
- [Configuration Change Requests on page 67](#)

Operational Requests

Operational requests are requests for information about the status of a device running Junos OS. Operational requests correspond to the Junos OS CLI operational mode commands. The Junos XML API defines a request tag element for many CLI commands. For example, the `<get-interface-information>` tag element corresponds to the

show interfaces command, and the **<get-chassis-inventory>** tag element requests the same information as the **show chassis hardware** command.

The following RPC requests detailed information about interface ge-2/3/0:

```
<rpc>
  <get-interface-information>
    <interface-name>ge-2/3/0</interface-name>
    <detail/>
  </get-interface-information>
</rpc>
]]>]]>
```

For more information, see [“Requesting Operational Information Using NETCONF” on page 175](#). For information about the Junos XML request tag elements available in the current Junos OS Release, see the *Junos XML API Operational Developer Reference*.

Configuration Information Requests

Configuration information requests are requests for information about the device's candidate configuration, a private configuration, or the committed configuration (the one currently in active use on the switching, routing, or security platform). The candidate and committed configurations diverge when there are uncommitted changes to the candidate configuration.

The NETCONF protocol defines the **<get-config>** operation for retrieving configuration information. The Junos XML API defines a tag element for every container and leaf statement in the configuration hierarchy.

The following example shows how to request information from the **[edit system login]** hierarchy level of the candidate configuration:

```
<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter type="subtree">
      <configuration>
        <system>
          <login/>
        </system>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For more information, see [“Requesting Configuration Data Using NETCONF” on page 184](#). For a summary of the available configuration tag elements, see the *Junos XML API Configuration Developer Reference*.

Configuration Change Requests

Configuration change requests are requests to change the candidate configuration, or to commit those changes to put them into active use on the device running Junos OS. The NETCONF protocol defines the `<edit-config>` and `<copy-config>` operations for changing configuration information. The Junos XML API defines a tag element for every CLI configuration statement described in the Junos OS configuration guides.

The following example shows how to create a new Junos OS user account called **admin** at the `[edit system login]` hierarchy level in the candidate configuration:

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <configuration>
        <system>
          <login>
            <user>
              <name>admin</name>
              <full-name>Administrator</full-name>
              <class>superuser</class>
            </user>
          </login>
        </system>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>
```

For more information about changing the candidate configuration, see [“Editing the Candidate Configuration Using NETCONF” on page 83](#). For a summary of Junos XML configuration tag elements, see the *Junos XML API Configuration Developer Reference*.

Related Documentation

- [Generating Well-Formed XML Documents on page 24](#)
- [Parsing the NETCONF Server Response on page 67](#)
- [XML and NETCONF XML Management Protocol Conventions Overview on page 10](#)
- [<rpc> on page 140](#)

Parsing the NETCONF Server Response

In a NETCONF session with a device running Junos OS, a client application sends RPCs to the NETCONF server to request information from and manage the configuration on the device. The NETCONF server encloses its response to each client request in a separate pair of opening `<rpc-reply>` and closing `</rpc-reply>` tags. Each response constitutes a well-formed XML document.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" \
  xmlns:junos="http://xml.juniper.net/junos/release/junos" \
  [echoed attributes]>
  <!-- tag elements representing a response -->
</rpc-reply>
]]>]]>
```

The **xmlns** attribute in the opening **<rpc-reply>** tag defines the namespace for enclosed tag elements that do not have the **junos:** prefix in their names and that are not enclosed in a child container tag that has the **xmlns** attribute with a different value.



NOTE: If you configure the **rfc-compliant** statement on the device, the NETCONF server explicitly declares the NETCONF namespace, which is bound to the **nc** prefix, and qualifies all NETCONF tags in its replies with the prefix.

The **xmlns:junos** attribute defines the default namespace for enclosed Junos XML tag elements that are qualified by the **junos:** prefix. The *release* variable in the URI represents the Junos OS release that is running on the NETCONF server device, for example 15.1R1.

Client applications must include code for parsing the stream of response tag elements coming from the NETCONF server, either processing them as they arrive or storing them until the response is complete. The NETCONF server returns three classes of responses:

- [Operational Responses on page 68](#)
- [Configuration Information Responses on page 69](#)
- [Configuration Change Responses on page 69](#)

Operational Responses

Operational responses are responses to requests for information about the status of a switching, routing, or security platform. They correspond to the output from CLI operational commands.

The Junos XML API defines response tag elements for all defined operational request tag elements. For example, the NETCONF server returns the information requested by the **<get-interface-information>** tag in a response tag element called **<interface-information>**, and returns the information requested by the **<get-chassis-inventory>** tag in a response tag called **<chassis-inventory>**. Operational responses also can be returned in formatted ASCII, which is enclosed in an **<output>** element. For more information about formatting operational responses, see [“Specifying the Output Format for Operational Information Requests in a NETCONF Session” on page 177](#).

The following sample response includes information about the interface ge-2/3/0. The namespace indicated by the **xmlns** attribute in the opening **<interface-information>** tag is for interface information for Junos OS Release 15.1. The opening tags appear on two lines here for legibility only:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" \
  xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
```



```

<interface-information \
  xmlns="http://xml.juniper.net/junos/15.1R1/junos-interface">
  <physical-interface>
    <name>ge-2/3/0</name>
    <!-- other data tag elements for the ge-2/3/0 interface - ->
  </physical-interface>
</interface-information>
</rpc-reply>
]]>]]>

```

For more information about the `xmlns` attribute and the contents of operational response tag elements, see [“Requesting Operational Information Using NETCONF” on page 175](#). For a summary of operational response tag elements, see the *Junos XML API Operational Developer Reference*.

Configuration Information Responses

Configuration information responses are responses to requests for information about the device's current configuration. The Junos XML API defines a tag element for every container and leaf statement in the configuration hierarchy.

The following sample response includes the information at the **[edit system login]** hierarchy level in the configuration hierarchy. For brevity, the sample shows only one user defined at this level. The opening `<rpc-reply>` tag appears on two lines for legibility only. For information about the attributes in the opening `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

```

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" \
  xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
  <data>
    <configuration attributes>
      <system>
        <login>
          <user>
            <name>admin</name>
            <full-name>Administrator</full-name>
            <!-- other data tag elements for the admin user -->
          </user>
        </login>
      </system>
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

Configuration Change Responses

Configuration change responses are responses to requests that change the state or contents of the device configuration. The NETCONF server indicates successful execution of a request by returning the `<ok/>` tag within the `<rpc-reply>` tag element:

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

If the operation fails, the `<rpc-reply>` tag element instead encloses an `<rpc-error>` element that describes the cause of the failure. For information about handling errors, see [“Handling an Error or Warning in a NETCONF Session” on page 70](#).

Related Documentation

- [Using a Standard API to Parse Response Tag Elements in NETCONF and Junos XML Protocol Sessions on page 70](#)
- [Handling an Error or Warning in a NETCONF Session on page 70](#)
- [Configuring RFC-Compliant NETCONF Sessions on page 79](#)
- [XML and NETCONF XML Management Protocol Conventions Overview on page 10](#)
- [Generating Well-Formed XML Documents on page 24](#)
- `<rpc-error>` on page 140

Using a Standard API to Parse Response Tag Elements in NETCONF and Junos XML Protocol Sessions

In a NETCONF or Junos XML protocol session, client applications can handle incoming XML tag elements by feeding them to a parser that is based on a standard API such as the Document Object Model (DOM) or Simple API for XML (SAX). Describing how to implement and use a parser is beyond the scope of this documentation.

Routines in the DOM accept incoming XML and build a tag hierarchy in the client application's memory. There are also DOM routines for manipulating an existing hierarchy. DOM implementations are available for several programming languages, including C, C++, Perl, and Java. For detailed information, see the *Document Object Model (DOM) Level 1 Specification* from the World Wide Web Consortium (W3C) at <http://www.w3.org/TR/REC-DOM-Level-1/>. Additional information is available from the Comprehensive Perl Archive Network (CPAN) at <http://search.cpan.org/~tjmathier/XML-DOM/lib/XML/DOM.pm>.

One potential drawback with DOM is that it always builds a hierarchy of tag elements, which can become very large. If a client application needs to handle only one subhierarchy at a time, it can use a parser that implements SAX instead. SAX accepts XML and feeds the tag elements directly to the client application, which must build its own tag hierarchy. For more information, see the official SAX website at <http://sax.sourceforge.net/>.

Related Documentation

- [Parsing the Junos XML Protocol Server Response](#)
- [Parsing the NETCONF Server Response on page 67](#)

Handling an Error or Warning in a NETCONF Session

In a NETCONF session with a device running Junos OS, a client application sends RPCs to the NETCONF server to request information from and manage the configuration on the device. The NETCONF server sends a response to each client request. If the server encounters an error condition, it emits an `<rpc-error>` element containing child elements that describe the error.

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rpc-error>
    <error-severity>error-severity</error-severity>
    <error-path>error-path</error-path>
    <error-message>error-message</error-message>
    <error-info>
      <bad-element>command-or-statement</bad-element>
    </error-info>
  </rpc-error>
</rpc-reply>
]]>]]>

```

<bad-element> identifies the command or configuration statement that was being processed when the error or warning occurred. For a configuration statement, the **<error-path>** tag element enclosed in the **<rpc-error>** tag element specifies the statement's parent hierarchy level.

<error-message> describes the error or warning in a natural-language text string.

<error-path> specifies the path to the Junos OS configuration hierarchy level at which the error or warning occurred, in the form of the CLI configuration mode banner.

<error-severity> indicates the severity of the event that caused the NETCONF server to return the **<rpc-error>** tag element. The two possible values are **error** and **warning**.

An error can occur while the server is performing any of the following operations, and the server can send a different combination of child tag elements in each case:

- Processing an operational request submitted by a client application
- Opening, locking, changing, committing, or closing a configuration as requested by a client application
- Parsing configuration data submitted by a client application in an **<edit-config>** tag element

Client applications must be prepared to receive and handle an **<rpc-error>** tag element at any time. The information in any response tag elements already received and related to the current request might be incomplete. The client application can include logic for deciding whether to discard or retain the information.

When the **<error-severity>** tag element has the value **error**, the usual response is for the client application to discard the information and terminate. When the **<error-severity>** tag element has the value **warning**, indicating that the problem is less serious, the usual response is for the client application to log the warning or pass it to the user and to continue parsing the server's response.

Related Documentation

- [Parsing the NETCONF Server Response on page 67](#)
- [<rpc-error> on page 140](#)

Locking and Unlocking the Candidate Configuration Using NETCONF

When a client application is requesting or changing configuration information, it can use one of the following methods to access the configuration:

- Lock the candidate configuration, which prevents other users or applications from changing it until the application releases the lock. This is equivalent to the CLI **configure exclusive** command.
- Change the candidate configuration without locking it. We do not recommend this method, because of the potential for conflicts with changes made by other applications or users that are editing the configuration at the same time.

If an application is simply requesting configuration information and not changing it, locking the configuration is not required. The application can begin requesting information immediately. However, if it is important that the information being returned not change during the session, it is appropriate to lock the configuration.

For more information about locking and unlocking the candidate configuration, see the following sections:

- [Locking the Candidate Configuration on page 72](#)
- [Unlocking the Candidate Configuration on page 73](#)

Locking the Candidate Configuration

To lock the candidate configuration, a client application emits the **<lock>** and **<target>** tag elements and the **<candidate/>** tag in the **<rpc>** tag element:

```
<rpc>
  <lock>
    <target>
      <candidate/>
    </target>
  </lock>
</rpc>
]]>]]>
```

Locking the candidate configuration prevents other users or applications from changing the candidate configuration until the lock is released. This is equivalent to the CLI **configure exclusive** command. Locking the configuration before making changes is recommended, particularly on devices where multiple users are authorized to change the configuration. A commit operation applies to all changes in the candidate configuration, not just those made by the user or application that requests the commit. Allowing multiple users or applications to make changes simultaneously can lead to unexpected results.

The NETCONF server confirms that it has locked the candidate by returning the **<ok/>** tag in the **<rpc-reply>** tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
```

```
</rpc-reply>
]]>]]>
```

If the NETCONF server cannot lock the configuration, the **<rpc-reply>** tag element instead encloses an **<rpc-error>** tag element explaining the reason for the failure. Reasons for the failure can include the following:

- Another user or application has already locked the candidate configuration. The error message reports the NETCONF session identifier of the user or application. If the client application has the necessary Junos access privilege, it can terminate the session that holds the lock. For more information, see [“Terminating a NETCONF Session” on page 74](#).
- The candidate configuration already includes changes that have not yet been committed. To commit the changes, see [“Committing the Candidate Configuration Using NETCONF” on page 114](#). To discard uncommitted changes, see [“Rolling Back the Candidate Configuration Using NETCONF” on page 95](#).

Only one application can hold the lock on the candidate configuration at a time. Other users and applications can read the candidate configuration while it is locked. The lock persists until either the NETCONF session ends or the client application unlocks the configuration by emitting the **<unlock>** tag element, as described in [“Unlocking the Candidate Configuration” on page 73](#).

If the candidate configuration is not committed before the client application unlocks it, or if the NETCONF session ends for any reason before the changes are committed, the changes are automatically discarded. The candidate and committed configurations remain unchanged.

Unlocking the Candidate Configuration

As long as a client application holds a lock on the candidate configuration, other applications and users cannot change the candidate. To unlock the candidate configuration, the client application includes the **<unlock>** and **<target>** tag elements and the **<candidate/>** tag in an **<rpc>** tag element:

```
<rpc>
  <unlock>
    <target>
      <candidate/>
    </target>
  </unlock>
</rpc>
]]>]]>
```

The NETCONF server confirms that it has unlocked the candidate by returning the **<ok/>** tag in the **<rpc-reply>** tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

If the NETCONF server cannot unlock the configuration, the **<rpc-reply>** tag element instead encloses an **<rpc-error>** tag element explaining the reason for the failure.

- Related Documentation**
- [Understanding the Client Application's Role in a NETCONF Session on page 24](#)
 - [<lock> on page 134](#)
 - [<target> on page 141](#)
 - [<unlock> on page 135](#)

Terminating a NETCONF Session

In a NETCONF session, a client application's attempt to lock the candidate configuration can fail because another user or application already holds the lock. In this case, the NETCONF server returns an error message that includes the username and process ID (PID) for the entity that holds the existing lock:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rpc-error>
    <error-severity>error</error-severity>
    <error-message>
      configuration database locked by:
      user terminal (pid PID) on since YYYY-MM-DD hh:mm:ss TZ, idle hh:mm:ss
      exclusive
    </error-message>
  </rpc-error>
</rpc-reply>
]]>]]>
```

If the client application has the Junos OS **maintenance** permission, it can end the session that holds the lock by emitting the **<kill-session>** and **<session-id>** tag elements in an **<rpc>** tag element. The **<session-id>** element specifies the PID obtained from the error message:

```
<rpc>
  <kill-session>
    <session-id>PID</session-id>
  </kill-session>
</rpc>
]]>]]>
```

The NETCONF server confirms that it has terminated the other session by returning the **<ok/>** tag in the **<rpc-reply>** tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

We recommend that the application include logic for determining whether it is appropriate to terminate another session, based on factors such as the identity of the user or application that holds the lock, or the length of idle time.

When a session is terminated, the NETCONF server that is servicing the session rolls back all uncommitted changes that have been made during the session. If a confirmed commit is pending (changes have been committed but not yet confirmed), the NETCONF server restores the configuration to its state before the confirmed commit instruction was issued.

For information about the confirmed commit operation, see [“Committing the Candidate Configuration Only After Confirmation Using NETCONF”](#) on page 114.

The following example shows how to terminate another session:

Client Application	NETCONF Server
<pre><rpc> <kill-session> <session-id>3250</session-id> </kill-session> </rpc>]]>]]></pre>	<pre><rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]></pre>

T2101

- Related Documentation**
- [Ending a NETCONF Session and Closing the Connection](#) on page 75
 - [Locking and Unlocking the Candidate Configuration Using NETCONF](#) on page 72
 - [<kill-session>](#) on page 134

Ending a NETCONF Session and Closing the Connection

When a client application is finished making requests, it ends the NETCONF session by emitting the empty `<close-session/>` tag within an `<rpc>` tag element:

```
<rpc>
  <close-session/>
</rpc>
]]>]]>
```

In response, the NETCONF server emits the `<ok/>` tag enclosed in an `<rpc-reply>` tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

Because the connection to the NETCONF server is an SSH subsystem, it closes automatically when the NETCONF session ends.

- Related Documentation**
- [<close-session/>](#) on page 127

Sample NETCONF Session

The following sections describe the sequence of tag elements in a sample NETCONF session with a device running Junos OS. The client application begins by establishing a connection to a NETCONF server.

- [Exchanging Initialization Tag Elements on page 76](#)
- [Sending an Operational Request on page 77](#)
- [Locking the Configuration on page 77](#)
- [Changing the Configuration on page 78](#)
- [Committing the Configuration on page 78](#)
- [Unlocking the Configuration on page 78](#)
- [Closing the NETCONF Session on page 79](#)

Exchanging Initialization Tag Elements

After the client application establishes a connection to a NETCONF server, the two exchange **<hello>** tag elements, as shown in the following example. For legibility, the example places the client application's **<hello>** tag element below the NETCONF server's. The two parties can actually emit their **<hello>** tag elements at the same time. For information about the **]]>]]>** character sequence used in this and the following examples, see [“Generating Well-Formed XML Documents” on page 24](#). For a detailed discussion of the **<hello>** tag element, see [“Exchanging <hello> Tag Elements” on page 61](#).

NETCONF Client Application Server

```
<hello>
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file </capability>
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
  </capabilities>
  <session-id>3911</session-id>
</hello>
]]>]]>

  <hello>
    <capabilities>
      <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
      <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>
      <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0</capability>
      <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>
      <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file</capability>
      <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    </capabilities>
  </hello>
]]>]]>
```

T2102

Sending an Operational Request

The client application emits the `<get-chassis-inventory>` tag element to request information about the device's chassis hardware. The NETCONF server returns the requested information in the `<chassis-inventory>` tag element.

Client Application	NETCONF Server
<pre> <rpc> <get-chassis-inventory> <detail/> </get-chassis-inventory> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <chassis-inventory xmlns="URL"> <chassis> <name>Chassis</name> <serial-number>1122</serial-number> <description>M320</description> <chassis-module> <name>Midplane</name> <!-- other child tags for the midplane --> </chassis-module> <!-- tags for other chassis modules --> </chassis> </chassis-inventory> </rpc-reply>]]>]]> </pre>

T2103

Locking the Configuration

The client application then prepares to incorporate a change into the candidate configuration by emitting the `<lock/>` tag to prevent any other users or applications from altering the candidate configuration at the same time. To confirm that the candidate configuration is locked, the NETCONF server returns an `<ok/>` tag in an `<rpc-reply>` tag element. For more information about locking the configuration, see [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#).

Client Application	NETCONF Server
<pre> <rpc> <lock> <target> <candidate/> </target> </lock> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>

T2104

Changing the Configuration

The client application now emits tag elements to create a new Junos OS login class called **network-mgmt** at the **[edit system login class]** hierarchy level in the candidate configuration. To confirm that the load operation was successful, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag element.

Client Application	NETCONF Server
<pre> <rpc> <edit-config> <target> <candidate/> </target> <config> <configuration> <system> <login> <class> <name>network-mgmt</name> <permissions>configure</permissions> <permissions>snmp</permissions> <permissions>system</permissions> </class> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>

T2105

Committing the Configuration

The client application then commits the candidate configuration. To confirm that the commit operation was successful, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag element. For more information about the commit operation, see [“Committing the Candidate Configuration Using NETCONF” on page 114](#).

Client Application	NETCONF Server
<pre> <rpc> <commit/> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>

T2106

Unlocking the Configuration

The client application unlocks (and by implication closes) the candidate configuration. To confirm that the unlock operation was successful, the NETCONF server returns an

`<ok/>` tag in an `<rpc-reply>` tag element. For more information about unlocking a configuration, see [“Locking and Unlocking the Candidate Configuration Using NETCONF” on page 72](#).

Client Application NETCONF Server

```

Client Application:
<rpc>
  <unlock>
    <target>
      <candidate/>
    </target>
  </unlock>
</rpc>
]]>]]>

NETCONF Server:
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

T2107

Closing the NETCONF Session

The client application closes the NETCONF session by emitting the `<close-session>` tag. For more information about closing the session, see [“Ending a NETCONF Session and Closing the Connection” on page 75](#).

Client Application NETCONF Server

```

Client Application:
<rpc>
  <close-session/>
</rpc>
]]>]]>

NETCONF Server:
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

T2108

Related Documentation

- [Generating Well-Formed XML Documents on page 24](#)
- [Starting the NETCONF Session on page 61](#)
- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)
- [Ending a NETCONF Session and Closing the Connection on page 75](#)

Configuring RFC-Compliant NETCONF Sessions

When using NETCONF to manage devices running Junos OS, you can require that the NETCONF server enforce certain behaviors during the NETCONF session by configuring the `rfc-compliant` statement at the `[edit system services netconf]` hierarchy level. Configuring the `rfc-compliant` statement affects the following aspects of the NETCONF session:

- Namespaces emitted in server replies
- `<get>` and `<get-config>` elements in RPC replies in cases where there is no configuration data to return

The differences are described in detail in the following sections:

- [Namespaces on page 80](#)
- [Changes to <get> and <get-config> Operations on page 81](#)

Namespaces

In a NETCONF session with a device running Junos OS, the NETCONF server, by default, sets the default namespace to the NETCONF namespace in the opening tag of the server's reply, and NETCONF tags are not qualified. For example:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    ...
  </capabilities>
  <session-id>27700</session-id>
</hello>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
```

When the **rfc-compliant** statement is configured on the device, the NETCONF server does not define a default namespace in its replies. Instead, the server includes a namespace declaration for the NETCONF namespace, which is bound to the **nc** prefix, and qualifies all NETCONF tags in its replies with the prefix. If you set the default namespace to the NETCONF namespace in an RPC request, the server discards the default namespace and emits its reply using only the declared namespace that is bound to the **nc** prefix.

The following sample output shows the NETCONF server's **<hello>** message and capabilities exchange when the **rfc-compliant** statement is configured. The **<hello>** tag contains the **xmlns:nc** declaration, and all NETCONF tags include the **nc** prefix.

```
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
    <nc:capability>urn:ietf:params:netconf:base:1.0</nc:capability>
    ...
  </nc:capabilities>
  <nc:session-id>27703</nc:session-id>
</nc:hello>
```

The following output shows a sample RPC reply when the **rfc-compliant** statement is configured:

```
<nc:rpc-reply
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
  <nc:data>
    <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm"
      junos:changed-seconds="1417554471"
      junos:changed-localtime="2014-12-02 13:07:51 PST">
      <!--configuration data-->
    </configuration>
    <database-status-information>
      <database-status>
        <user>root</user>
      <terminal></terminal>
```

```

    <pid>47868</pid>
    <start-time junos:seconds="1417560303">2014-12-02 14:45:03 PST</start-time>

    <edit-path></edit-path>
    </database-status>
  </database-status-information>
</nc:data>
</nc:rpc-reply>

```

Changes to <get> and <get-config> Operations

The **rfc-compliant** statement affects the <get> and <get-config> server replies only in cases where there is no configuration data to return. This can occur, for example, when you apply a filter to return a subset of the configuration, and that portion of the configuration is empty.

If you execute the <get> or <get-config> operation, and there is no configuration data in the requested hierarchy, then if the **rfc-compliant** statement is not configured, the RPC reply contains an empty <configuration> tag inside the <data> element.

```

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/15.1D0/junos">
  <data>
    <configuration>
  </configuration>
  </data>
</rpc-reply>

```

If you execute the <get> or <get-config> operation, and there is no configuration data in the requested hierarchy, then if the **rfc-compliant** statement is configured, the RPC reply does not include the empty <configuration> tag inside the <data> element.

```

<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
  <nc:data>
</nc:data>
</nc:rpc-reply>

```

Related Documentation • [rfc-compliant on page 122](#)

CHAPTER 10

Changing the Configuration

- [Editing the Candidate Configuration Using NETCONF on page 83](#)
- [Uploading and Formatting Configuration Data in a NETCONF Session on page 85](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)
- [Handling Errors While Editing the Candidate Configuration in a NETCONF Session on page 93](#)
- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
- [Rolling Back the Candidate Configuration Using NETCONF on page 95](#)
- [Deleting the Candidate Configuration Using NETCONF on page 96](#)
- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Merging Configuration Elements Using NETCONF on page 98](#)
- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Replacing Patterns in Configuration Data Using the NETCONF or Junos XML Protocol on page 108](#)

Editing the Candidate Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, you can use NETCONF XML management protocol operations along with Junos XML or command-line interface (CLI) configuration statements to change the configuration on a routing, switching, or security platform. The NETCONF protocol operations **<copy-config>**, **<edit-config>**, and **<discard-changes>** offer functionality that is analogous to configuration mode commands in the Junos OS CLI. The Junos XML tag elements described here correspond to configuration statements, which are described in the Junos OS configuration guides.

To change the candidate configuration on a device, a client application emits the **<copy-config>**, the **<edit-config>**, or the **<discard-changes>** tag element and the corresponding tag subelements within the **<rpc>** tag element.

The following examples shows the various tag elements available:

```
<rpc>
  <copy-config>
```

```
<target><candidate/></target>
<error-operation> (ignore-error | stop-on-error) </error-operation>
<source><url>location</url></source>
</copy-config>
</rpc>
]]>]]>

<rpc>
  <edit-config>
    <target><candidate/></target>
    <default-operation>operation</default-operation>
    <error-operation>error</error-operation>
    <(config | config-text | url)>
      <!-- configuration change file or data -->
    </(config | config-text | url)>
  </edit-config>
</rpc>
]]>]]>

<rpc>
  <discard-changes/>
</rpc>
]]>]]>
```

The three tags—**<copy-config>**, **<edit-config>**, and **<discard-changes>**—correspond to the three basic configuration tasks available to you:

- Overwriting the candidate configuration with a new configuration—Using the **<copy-config>** tag element, you can replace the current candidate configuration with a new configuration.
- Editing the candidate configuration elements—Using the **<edit-config>** tag element, you can add, change, or delete specific configuration elements within the candidate configuration. To specify how the device should handle configuration changes, see [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#).
- Rolling back changes to the current configuration—Using the **<discard-changes>** tag element, you can roll back the candidate configuration to a previously committed configuration. This tag element provides functionality analogous to the CLI command **rollback**.

Related Documentation

- [Uploading and Formatting Configuration Data in a NETCONF Session on page 85](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)
- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
- [Rolling Back the Candidate Configuration Using NETCONF on page 95](#)
- [Understanding the Client Application's Role in a NETCONF Session on page 24](#)
- [<copy-config> on page 128](#)
- [<discard-changes/> on page 129](#)
- [<edit-config> on page 129](#)

Uploading and Formatting Configuration Data in a NETCONF Session

In a NETCONF session with a device running Junos OS, a client application can use a text file or streaming data to upload configuration data to the candidate configuration. The data delivered can be in one of two formats: Junos XML or CLI configuration statements. You can specify the delivery mechanism and the format used when delivering configuration changes to the device.

When formatting your configuration data output, you can choose to stream your configuration changes within your session or reference data files that include the desired configuration changes. Each method has advantages and disadvantages. Streaming data allows you to send your configuration change data in line, using your NETCONF connection. This is useful when the device is behind a firewall and you cannot establish another connection to upload a data file. With text files you can keep the edit configuration commands simple; with data files, there is no need to include the possibly complex configuration data stream.

You can format the configuration data using one of two formats: Junos XML or CLI configuration statements. The choice between one data format over the other is personal preference.

The delivery mechanism and the format are discussed in detail in the following sections:

- [Referencing Configuration Data Files on page 85](#)
- [Streaming Configuration Data on page 87](#)
- [Formatting Data: Junos XML versus CLI Configuration Statements on page 88](#)

Referencing Configuration Data Files

To reference configuration data as a file, a client application emits the file location between `<url>` tag elements within the `<rpc>` and the `<edit-config>` or `<copy-config>` tag elements.

```
<rpc>
  <copy-config>
    <target>
      <candidate/>
    </target>
    <source>
      <url>
        <!-- location and name of file containing configuration data -->
      </url>
    </source>
  </copy-config>
</rpc>
]]>]]>

<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
```

```
<url>
  <!-- location and name of file containing configuration data -->
</url>
</edit-config>
</rpc>
]]>]]>
```

The data within these files can be formatted as either Junos XML or CLI configuration statements. When the configuration data is formatted as CLI configuration statements, you set the `<url>` format attribute to **text**.

```
<url format="text">
  <!-- location and name of file containing configuration data -->
</url>
```

The configuration file can be placed locally or as a network resource:

- When placed locally, the configuration file path can be relative or absolute:
 - Relative file path—The file location is based on the user's home directory.
 - Absolute file path—The file location is based on the directory structure of the device, for example `<drive>:filename` or `<drive>:/path/filename`. If you are using removable media, the drive can be in the MS-DOS or UNIX (UFS) format.
- When located on the network, the configuration file can be accessed using FTP or HTTP:
- FTP example:

```
ftp://username:password@hostname/path/filename
```



NOTE: The default value for the FTP *path* variable is the user's home directory. Thus, by default the file path to the configuration file is relative to the user directory. To specify an absolute path when using FTP, start the path with the characters `%2F`; for example:
`ftp://username:password@hostname/%2Fpath/filename.`

- HTTP example:

```
http://username:password@hostname/path/filename
```

Before loading the file, the client application or an administrator saves Junos XML tag elements or CLI configuration statements as the contents of the file. The file includes the tag elements or configuration statements representing all levels of the configuration hierarchy from the root (represented by the `<configuration>` tag element) down to each element to change. The notation is the same as that used to request configuration information. For more detailed information about the Junos XML representation of Junos configuration statements, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

The following example shows how to incorporate configuration data stored in the file `/var/tmp/configFile` on the FTP server called `ftp.myco.com`:

Client Application	NETCONF Server	
<pre> <rpc message-id="messageID"> <edit-config> <target> <candidate/> </target> <url> ftp://admin:AdminPwd@ftp.myco.com/%F2var/tmp/configFile </url> </edit-config> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>	T2134

Streaming Configuration Data

To provide configuration data as a data stream, a client application emits the `<config>` or `<config-text>` tag elements within the `<rpc>` and `<edit-config>` tag elements. To specify the configuration elements to change, the application emits Junos XML or CLI configuration statements representing all levels of the configuration hierarchy from the root (represented by the `<configuration>` or `<configuration-text>` tag element) down to each element to change. The Junos XML notation is the same as that used to request configuration information. For more detailed information about the mappings between Junos configuration elements and Junos XML tag elements, see ["Mapping Configuration Statements to Junos XML Tag Elements" on page 15](#). The CLI configuration statement notation are further described in the *CLI User Guide*.

```

<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config> or <config-text>
      <configuration> or <configuration-text>
        <!-- configuration changes -->
      </configuration> or </configuration-text>
    </config> or </config-text>
  </edit-config>
</rpc>
]]>]]>

```

The following example shows how to provide Junos XML configuration data for the `messages` system log file in a data stream:

Client Application NETCONF Server

```

<rpc message-id="messageID">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <configuration>
        <system>
          <syslog>
            <file>
              <name>messages</name>
              <contents>
                <name>any</name>
                <warning/>
              </contents>
              <contents>
                <name>authorization</name>
                <info/>
              </contents>
            </file>
          </syslog>
        </system>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>

<rpc-reply xmlns="URN" xmlns:junos=" URL">
  <ok/>
</rpc-reply>
]]>]]>

```

T2135

Formatting Data: Junos XML versus CLI Configuration Statements

You can format the configuration data using one of two formats: Junos XML or CLI configuration statements. The choice between one data format over the other is personal preference.

If you are supplying the configuration changes in the form of data files, you enclose the data filename and path within `<url>` tags. By default, these tags specify that the referenced data files are written in Junos XML. Thus, the following code declares that the data within the file is Junos XML:

```
<url>dataFile</url>
```

To specify that the data file be written as CLI configuration statements, you set the `<url>` tag's format attribute to **text**:

```
<url format="text">dataFile</url>
```

When streaming data, you specify the data format by selecting one of two tags: `<config>` for Junos XML statements and `<config-text>` for CLI configuration statements.

In the following example, Junos XML formatted configuration data is included between the `<configuration>` tag:

```
<config>
  <configuration>
    <system>
      <services>
        <ssh>
          <protocol-version>v2</protocol-version>
        </ssh>
      </services>
    </system>
  </configuration>
</config>
```

In this next example, the same data is formatted as CLI configuration statements and included within `<configuration-text>` tags:

```
<config-text>
  <configuration-text>
    system {
      services {
        ssh {
          protocol-version v2 ;
        }
      }
    }
  </configuration-text>
</config-text>
```

**Related
Documentation**

- [Editing the Candidate Configuration Using NETCONF on page 83](#)
- [XML Overview on page 9](#)
- [<copy-config> on page 128](#)
- [<edit-config> on page 129](#)

Setting the Edit Configuration Mode in a NETCONF Session

When sending operation data to the NETCONF server, you can specify how a device should handle these configuration changes. This is known as the edit configuration mode. You can set the edit configuration mode globally for the entire session. You can also set the edit mode only for specific elements within the session.

The device has the following edit configuration modes:

- **merge**—The device merges new configuration data into the current candidate configuration. This is the default.
- **replace**—The device replaces existing configuration data with the new configuration data.
- **no-change**—The device does not change the existing configuration unless the new configuration element includes an operation attribute.

To set the mode globally for the session, place a configuration mode value within `<default-operation>` tags:

```
<rpc>
  <edit-config>
    <default-operation>mode</default-operation>
  </edit-config>
</rpc>
```

You can also set the mode for a specific configuration statement by adding an **operation** attribute with a value of **replace** to the configuration element:

```
<rpc>
  <edit-config>
    <config>
      <configuration>
        <protocols>
          <rip>
            <message-size operation="replace">255</message-size>
          </rip>
        </protocols>
      </configuration>
    </config>
  </edit-config>
</rpc>
```

You can set a global edit configuration mode for an entire set of configuration changes and specify a different mode for individual elements that you want handled in a different manner. For example:

```
<rpc>
  <edit-config>
    <default-operation>merge</default-operation>
    <config>
      <configuration>
        <protocols>
          <rip>
            <message-size operation="replace">255</message-size>
          </rip>
        </protocols>
      </configuration>
    </config>
  </edit-config>
</rpc>
```

```

    </protocols>
  </configuration>
</config>
</edit-config>
</rpc>

```

The edit configuration modes are discussed in more detail in the following sections:

- [Specifying the merge Data Mode on page 91](#)
- [Specifying the replace Data Mode on page 91](#)
- [Specifying the no-change Data Mode on page 92](#)

Specifying the merge Data Mode

By default, the NETCONF server *merges* new configuration data into the candidate configuration. Thus, if no edit-configuration mode is specified, the device will merge the new configuration elements into the existing candidate configuration. Merging configurations is performed according to the following rules:

- A configuration element (hierarchy level or configuration object) that exists in the candidate configuration but not in the new configuration remains unchanged.
- A configuration element that exists in the new configuration but not in the candidate configuration is added to the candidate configuration.
- If a configuration element exists in both configurations, the following results occur:
 - If a child statement of the configuration element (represented by a child tag element) exists in the candidate configuration but not in the new configuration, it remains unchanged.
 - If a child statement exists in the new configuration but not in the candidate, it is added to the candidate configuration.
 - If a child statement exists in both configurations, the value in the new data replaces the value in the candidate configuration.

To explicitly specify that data be merged, the application can include the **<default-operation>** tag element with the value **merge** in the **<edit-config>** tag element:

```

<rpc>
  <edit-config>
    <default-operation>merge</default-operation>
    <!-- other child tag elements of the <edit-config> tag element -->
  </edit-config>
</rpc>
]]>]]>

```

Specifying the replace Data Mode

In the *replace* edit configuration mode, the new configuration data completely replaces the candidate configuration. To specify that the data be replaced, the application can include the **<default-operation>** tag element with the value **replace** in the **<edit-config>** tag element:

```
<rpc>
  <edit-config>
    <default-operation>replace</default-operation>
  </edit-config>
</rpc>
]]>]]>
```

We recommend using the global replace mode only when you plan to completely overwrite the candidate configuration with new configuration data. Furthermore, when the edit configuration mode is set to **replace**, we do not recommend using the **operation** attribute on individual configuration elements.

You can also replace individual configuration elements while merging or creating others. See [“Replacing Configuration Elements Using NETCONF” on page 106](#).

Specifying the no-change Data Mode

In the *no-change* mode, configuration changes to the configuration are ignored. This mode is useful when you are deleting elements, and it prevents the NETCONF server from creating parent hierarchy levels for an element that is being deleted. For more information, see [“Deleting Configuration Elements Using NETCONF” on page 101](#).

To set the no-change edit configuration mode globally, the application can include the **<default-operation>** tag element with the value **none** in the **<edit-config>** tag element:

```
<rpc>
  <edit-config>
    <default-operation>none</default-operation>
  </edit-config>
</rpc>
```



NOTE: If the new configuration data includes a configuration element that does not exist in the candidate, the NETCONF server returns an error. We recommend using no-change mode only when removing configuration elements from the candidate configuration. When creating or modifying elements, applications need to use merge mode.

When the no-change edit configuration mode is set globally, using the **<default-operation>** tag, you can override this behavior by specifying a different edit configuration mode for a specific element using the **operation** attribute. For example:

```
<rpc>
  <edit-config>
    <default-operation>none</default-operation>
  <config>
    <configuration>
      <system>
        <services>
          <outbound-ssh>
            <client>
              <name>test</name>
              <device-id>test</device-id>
              <keep-alive>
```



```

        <retry operation="merge">4</retry>
        <timeout operation="merge">15</timeout>
    </keep-alive>
</client>
</outbound-ssh>
</services>
</system>
</configuration>
</config>
</edit-config>
</rpc>

```

Related Documentation • [Deleting Configuration Elements Using NETCONF on page 101](#)

Handling Errors While Editing the Candidate Configuration in a NETCONF Session

In a NETCONF session with a device running Junos OS, you can use NETCONF XML management protocol operations along with Junos XML or command-line interface (CLI) configuration statements to change the configuration on a routing, switching, or security platform. If the NETCONF server cannot incorporate the configuration data, the server returns the **<rpc-error>** tag element with information explaining the reason for the failure. By default, when the NETCONF server encounters an error while incorporating new configuration data into the candidate configuration, it halts the incorporation process. You can explicitly specify that the NETCONF server ignore errors or halt on error when incorporating new configuration data by including the **<error-option>** tag element.

A client application can explicitly specify that the NETCONF server stop incorporating new configuration data when it encounters an error. The application includes the **<error-option>** tag element with the value **stop-on-error** in the **<edit-config>** tag element:

```

<rpc>
  <edit-config>
    <error-option>stop-on-error</error-option>
    <!-- other child tag elements of the <edit-config> tag element -->
  </edit-config>
</rpc>
]]>]]>

```

Alternatively, the application can specify that the NETCONF server continue to incorporate new configuration data when it encounters an error. The application includes the **<error-option>** tag element with the value **ignore-error** in the **<edit-config>** tag element:

```

<rpc>
  <edit-config>
    <error-option>ignore-error</error-option>
    <!-- other child tag elements of the <edit-config> tag element -->
  </edit-config>
</rpc>
]]>]]>

```

The client application can include the optional **<test-option>** tag element described in the NETCONF specification. Regardless of the value provided, the NETCONF server for the Junos OS performs a basic syntax check on the configuration data in the **<edit-config>**

tag element. When the **<test-option>** tag is included, NETCONF performs a complete syntactic and semantic validation in response to the **<commit>** and **<validate>** tag elements (that is, when the configuration is committed or explicitly checked), but not in response to the **<edit-config>** tag element.

Related Documentation

- [Editing the Candidate Configuration Using NETCONF on page 83](#)
- [Verifying the Configuration Syntax Using NETCONF on page 113](#)
- [Committing the Candidate Configuration Using NETCONF on page 114](#)
- [Uploading and Formatting Configuration Data in a NETCONF Session on page 85](#)

Replacing the Candidate Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, you can replace the candidate configuration with a new configuration file using either the **<copy-config>** tag or the **<edit-config>** tag with the **<default-operation>** subtag value set to **replace**.

- [Using <copy-config> to Replace the Candidate Configuration on page 94](#)
- [Using <edit-config> to Replace the Candidate Configuration on page 94](#)

Using <copy-config> to Replace the Candidate Configuration

One method for replacing the entire candidate configuration is to include the **<copy-config>** tag element in the **<rpc>** element. The **<target>** tag encloses the **<candidate/>** tag to indicate that the new configuration data replaces the candidate configuration. The **<source>** tag element encloses the **<url>** tag element, which specifies the filename that contains the new configuration data. When the configuration data is formatted as Junos XML tag elements, set the **<url>** **format** attribute to **xml** or omit the attribute. When the configuration data is formatted as CLI configuration statements, set the **<url>** **format** attribute to **text**.

```
<rpc>
  <copy-config>
    <target>
      <candidate/>
    </target>
    <source>
      <url format="(xml | text)">
        <!-- location specifier for file containing the new configuration -->
      </url>
    </source>
  </copy-config>
</rpc>
]]>]]>
```

Using <edit-config> to Replace the Candidate Configuration

The other method for replacing the entire candidate configuration is to set the edit configuration mode to **replace** as a global variable. The candidate configuration includes the **<default-operation>** tag element with the value **replace** in the **<edit-config>** tag element, as described in ["Setting the Edit Configuration Mode in a NETCONF Session"](#)

on page 90. To specify the new configuration data, the application includes a `<config>` or `<config-text>` tag element that contains the data or a `<url>` tag element that names the file containing the data, as discussed in “Uploading and Formatting Configuration Data in a NETCONF Session” on page 85.

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <default-operation>replace</default-operation>

    <!-- EITHER -->
    <config>
      <configuration>
        <!-- tag elements representing the configuration elements to change -->
      </configuration>
    </config>
    <!-- OR -->
    <config-text>
      <configuration-text>
        <!-- configuration data in text format -->
      </configuration-text>
    </config-text>
    <!-- OR -->
    <url>
      <!-- location specifier for file containing changes -->
    </url>

  </edit-config>
</rpc>
]]>]]>
```

Related Documentation

- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Uploading and Formatting Configuration Data in a NETCONF Session on page 85](#)
- [<copy-config> on page 128](#)
- [<edit-config> on page 129](#)

Rolling Back the Candidate Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, you can roll back the candidate configuration to the current running configuration, which removes any uncommitted changes from the candidate configuration. This operation is equivalent to the CLI configuration mode **rollback 0** command.

To roll back the candidate configuration to the current running configuration, enclose the `<discard-changes>` tag within the `<rpc>` element.

```
<rpc>
  <discard-changes/>
```

```
</rpc>  
]]>]]>
```

After you issue the `</discard-changes>` tag, the NETCONF server indicates that it successfully discarded the changes by returning the `<ok/>` tag.

**Related
Documentation**

- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
- [Requesting a Previous \(Rollback\) Configuration Using NETCONF on page 204](#)
- [<discard-changes/> on page 129](#)

Deleting the Candidate Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, the `<delete-config>` tag element permits you to delete the current candidate configuration. Exercise caution when issuing the `<delete-config>` tag element. If you commit an empty candidate configuration, the device will go offline.

To delete the candidate configuration, insert the `<delete-config>` tag element in the `<rpc>` element. Specify the candidate configuration by enclosing the `<candidate/>` tag within the `<target>` tag.

```
<rpc>  
  <delete-config>  
    <target>  
      <candidate/>  
    </target>  
  </delete-config>  
</rpc>
```



WARNING: If you take the device offline, you will need to access the device through the console port on the device. From this console, you can access the CLI and perform a rollback to a suitable configuration. For more information on the console port, see the hardware manual for your specific device.

**Related
Documentation**

- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
- [Rolling Back the Candidate Configuration Using NETCONF on page 95](#)
- [<delete-config> on page 129](#)

Changing Individual Configuration Elements Using NETCONF

In a NETCONF session with a device running Junos OS, you change individual configuration elements within a candidate configuration using the `<edit-config>` tag element within the `<rpc>` tag. By default, the NETCONF server merges new configuration data into the existing candidate configuration. However, a client application can also replace, create,

or delete individual configuration elements (hierarchy levels or configuration objects). The same basic tag elements are emitted for all operations: `<config>`, `<config-text>`, or `<url>` tag sub-elements within the `<edit-config>` tag element:

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>

    <!-- EITHER -->
    <config>
      <configuration>
        <!-- tag elements representing the configuration elements to change -->
      </configuration>
    </config>
    <!-- OR -->
    <config-text>
      <configuration-text>
        <!-- configuration data in text format -->
      </configuration-text>
    </config-text>
    <!-- OR -->
    <url>
      <!-- location specifier for file containing changes -->
    </url>

  </edit-config>
</rpc>
]]>]]>
```

Using configuration data within the `<config>` or `<config-text>` tag elements or in the file specified by `<url>` tag element, the application defines a configuration element by including the tag elements representing all levels of the configuration hierarchy from the root down to the immediate parent level for the element. To represent the element, the application includes its container tag element. The child tag elements included within the container tag element depend on the operation.

For more information about the tag elements that represent configuration statements, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#). For information about the tag elements for a specific configuration element, see the *Junos XML API Configuration Developer Reference*.

The NETCONF server indicates that it changed the configuration in the requested way by enclosing the `<ok/>` tag in the `<rpc-reply>` tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

Related Documentation

- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Merging Configuration Elements Using NETCONF on page 98](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)

Merging Configuration Elements Using NETCONF

In a NETCONF session with a device running Junos OS, to merge configuration elements, including hierarchy levels or configuration objects, into the candidate configuration, a client application emits the basic tag elements described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#).

To represent each element to merge in (either within the `<config>` or `<config-text>` tag elements or in the file specified by the `<url>` tag element), the application includes the tag elements representing its parent hierarchy levels and its container tag element, as described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#). Within the container tag, the application includes each of the element’s identifier tag elements (if it has them) and the tag element for each child to add or for which to set a different value. In the following, the identifier tag element is called `<name>`:

```
<configuration>
  <!-- opening tags for each parent of the element -->
  <element>
    <name>identifier</name>
    <!-- - child tag elements to add or change -->
  </element>
  <!-- closing tags for each parent of the element -->
</configuration>
```

The NETCONF server merges in the new configuration element according to the rules specified in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). As described in that section, the application can explicitly specify merge mode by including the `<default-operation>` tag element with the value `merge` in the `<edit-config>` tag element.

The following example shows how to merge information for a new interface called so-3/0/0 into the `[edit interfaces]` hierarchy level in the candidate configuration:

Client Application

```

<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <configuration>
        <interfaces>
          <interface>
            <name>so-3/0/0</name>
            <unit>
              <name>0</name>
              <family>
                <inet>
                  <address>
                    <name>10.0.0.1/8</name>
                  </address>
                </inet>
              </family>
            </unit>
          </interface>
        </interfaces>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>

```

NETCONF Server

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

T2120

Related Documentation

- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)

Creating Configuration Elements Using NETCONF

In a NETCONF session with a device running Junos OS, to create configuration elements, including hierarchy levels or configuration objects, that do not already exist in the candidate configuration, a client application emits the basic tag elements described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#).

To represent each configuration element being created (either within the `<config>` or `<config-text>` tag elements or in the file specified by the `<url>` tag element), the application emits the tag elements representing its parent hierarchy levels and its container tag element, as described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#). Within the container tag, the application includes each of the element’s identifier tag elements (if it has them) and all child tag elements (with values,

if appropriate) that are being defined for the element. In the following, the identifier tag element is called `<name>`. The application includes the `operation="create"` attribute in the opening container tag:

```
<configuration>
  <!-- opening tags for each parent of the element -->
  <element operation="create">
    <name>identifier</name> <!-- if the element has an identifier -->
    <!-- other child tag elements -->
  </element>
  <!-- closing tags for each parent of the element -->
</configuration>
```

The NETCONF server adds the new element to the candidate configuration only if there is no existing element with that name (for a hierarchy level) or with the same identifiers (for a configuration object).

The following example shows how to enable OSPF on a device if it is not already configured:

Client Application

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <configuration>
        <protocols>
          <ospf operation="create">
            <area>
              <name>0</name>
              <interface>
                <name>at-0/1/0.100</name>
              </interface>
            </area>
          </ospf>
        </protocols>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>
```

NETCONF Server

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

T2122

Related Documentation

- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Merging Configuration Elements Using NETCONF on page 98](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)

Deleting Configuration Elements Using NETCONF

In a NETCONF session with a device running Junos OS, to delete a configuration element, including hierarchy levels or configuration objects, from the candidate configuration, a client application emits the basic tag elements described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#). It also emits the `<default-operation>` tag element with the value `none` to change the default mode to no-change.

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <default-operation>none</default-operation>

    <!-- EITHER -->
    <config>
      <configuration>
        <!-- tag elements representing the configuration elements to delete -->
      </configuration>
    </config>
  <!-- OR -->
  <url>
    <!-- location specifier for file containing elements to delete -->
  </url>

  </edit-config>
</rpc>
]]>]]>
```

In no-change mode, existing configuration elements remain unchanged unless the corresponding element in the new configuration has the `operation="delete"` attribute in its opening tag. This mode prevents the NETCONF server from creating parent hierarchy levels for an element that is being deleted. We recommend that the only operation performed in no-change mode be deletion. When merging, replacing, or creating configuration elements, client applications use merge mode.

To represent each configuration element being deleted (either within the `<config>` tag element or in the file named by the `<url>` tag element), the application emits the tag elements representing its parent hierarchy levels, as described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#). The tag element in which the `operation="delete"` attribute is included depends on the element type, as described in the following sections:

- [Deleting a Hierarchy Level or Container Object on page 102](#)
- [Deleting a Configuration Object That Has an Identifier on page 102](#)
- [Deleting a Single-Value or Fixed-Form Option from a Configuration Object on page 103](#)
- [Deleting Values from a Multi-value Option of a Configuration Object on page 104](#)

Deleting a Hierarchy Level or Container Object

To delete a hierarchy level and all of its children (or a container object that has children but no identifier), a client application includes the **operation="delete"** attribute in the empty tag that represents the level:

```
<configuration>
  <!-- opening tags for each parent level -->
  <level-to-delete operation="delete"/>
  <!-- closing tags for each parent level -->
</configuration>
```

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value **none**, as described in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). For more information about hierarchy levels and container objects, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

The following example shows how to remove the **[edit protocols ospf]** hierarchy level of the candidate configuration:

Client Application

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <default-operation>none</default-operation>
    <config>
      <configuration>
        <protocols>
          <ospf operation="delete"/>
        </protocols>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>
```

NETCONF Server

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

T2123

Deleting a Configuration Object That Has an Identifier

To delete a configuration object that has an identifier, a client application includes the **operation="delete"** attribute in the container tag element for the object. Inside the container tag element, it includes the identifier tag element only, not any tag elements that represent other characteristics. In the following, the identifier tag element is called **<name>**:

```
<configuration>
  <!-- opening tags for each parent of the object -->
  <object operation="delete">
```

```

    <name>identifier</name>
  </object>
  <!-- closing tags for each parent of the object -->
</configuration>

```



NOTE: The delete attribute appears in the opening container tag, not in the identifier tag element. The presence of the identifier tag element results in the removal of the specified object, not in the removal of the entire hierarchy level represented by the container tag element.

We recommend that the application set the default mode to no-change by including the `<default-operation>` tag element with the value `none`, as described in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). For more information about identifiers, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

The following example shows how to remove the user object **barbara** from the `[edit system login user]` hierarchy level in the candidate configuration:

Client Application

NETCONF Server

```

<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <default-operation>none</default-operation>
    <config>
      <configuration>
        <system>
          <login>
            <user operation="delete">
              <name>barbara</name>
            </user>
          </login>
        </system>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>

```

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

T2124

Deleting a Single-Value or Fixed-Form Option from a Configuration Object

To delete from a configuration object either a fixed-form option or an option that takes just one value, a client application includes the `operation="delete"` attribute in the tag element for the option. In the following, the identifier tag element for the object is called `<name>`. (For information about deleting an option that can take multiple values, see [“Deleting Values from a Multi-value Option of a Configuration Object” on page 104](#).)

```

<configuration>
  <!-- opening tags for each parent of the object -->
  <object>
    <name>identifier</name> <!-- if object has an identifier -->
    <option1 operation="delete">
    <option2 operation="delete">
    <!-- tag elements for other options to delete -->
  </object>
  <!-- closing tags for each parent of the object -->
</configuration>

```

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value **none**, as described in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). For more information about options, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

The following example shows how to remove the fixed-form **disable** option at the **[edit forwarding-options sampling]** hierarchy level:

Client Application	NETCONF Server
<pre> <rpc> <edit-config> <target> <candidate/> </target> <default-operation>none</default-operation> <config> <configuration> <forwarding-options> <sampling> <disable operation="delete"/> </sampling> </forwarding-options> </configuration> </config> </edit-config> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>

T2125

Deleting Values from a Multi-value Option of a Configuration Object

As described in [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#), some Junos OS configuration objects are leaf statements that have multiple values. In the formatted ASCII CLI representation, the values are enclosed in square brackets following the name of the object:

```
object[value1 value2 value3 ...];
```

The Junos XML representation does not use a parent tag for the object, but instead uses a separate instance of the object tag element for each value. In the following, the identifier tag element is called **<name>**:

```

<parent-object>
  <name>identifier</name>
  <object>value1</object>
  <object>value2</object>
  <object>value3</object>
</parent-object>

```

To remove one or more values for such an object, a client application includes the **operation="delete"** attribute in the opening tag for each value. It does not include tag elements that represent values to be retained. The identifier tag element in the following is called **<name>**:

```

<configuration>
  <!-- opening tags for each parent of the parent object -->
  <parent-object>
    <name>identifier</name>
    <object operation="delete">value1</object>
    <object operation="delete">value2</object>
  </parent-object>
  <!-- closing tags for each parent of the parent object -->
</configuration>

```

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value **none**, as described in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). For more information about leaf statements with multiple values, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

The following example shows how to remove two of the permissions granted to the **user-accounts** login class:

Client Application	NETCONF Server
<pre> <rpc> <edit-config> <target> <candidate/> </target> <default-operation>none</default-operation> <config> <configuration> <system> <login> <class> <name>user-accounts</name> <permissions operation="delete">configure</permissions> <permissions operation="delete">control</permissions> </class> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]> </pre>

T2126

Related Documentation

- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Deleting the Candidate Configuration Using NETCONF on page 96](#)
- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Merging Configuration Elements Using NETCONF on page 98](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)

Replacing Configuration Elements Using NETCONF

In a NETCONF session with a device running Junos OS, to replace configuration elements, including hierarchy levels or configuration objects, in the candidate configuration, a client application emits the basic tag elements described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#).

To represent the new definition for each configuration element being replaced (either within the **<config>** or **<config-text>** tag elements or in the file specified by the **<url>** tag element), the application emits the tag elements representing its parent hierarchy levels and its container tag element, as described in [“Changing Individual Configuration Elements Using NETCONF” on page 96](#). Within the container tag, the application includes each of the element’s identifier tag elements (if it has them) and all child tag elements (with values, if appropriate) that are being defined for the new version of the element. In the

following, the identifier tag element is called **<name>**. The application includes the **operation="replace"** attribute in the opening container tag:

```
<configuration>
  <!-- opening tags for each parent of the element -->
  <container-tag operation="replace">
    <name>identifier</name>
    <!-- other child tag elements -->
  </container-tag>
  <!-- closing tags for each parent of the element -->
</configuration>
```

The NETCONF server removes the existing element that has the specified identifiers and inserts the new element.

The application can also replace all objects in the configuration in one operation. For instructions, see [“Replacing the Candidate Configuration Using NETCONF” on page 94](#).

The following example shows how to grant new permissions for the object named **operator** at the **[edit system login class]** hierarchy level.

Client Application

```
<rpc>
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <configuration>
        <system>
          <login>
            <class operation="replace">
              <name>operator</name>
              <permissions>configure</permissions>
              <permissions>admin-control</permissions>
            </class>
          </login>
        </system>
      </configuration>
    </config>
  </edit-config>
</rpc>
]]>]]>
```

NETCONF Server

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

T2121

Related Documentation

- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Merging Configuration Elements Using NETCONF on page 98](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)

Replacing Patterns in Configuration Data Using the NETCONF or Junos XML Protocol

In a NETCONF or Junos XML protocol session with a device running Junos OS, you can replace variables and identifiers in the configuration by including the **replace-pattern** attribute when performing a **<load-configuration>** operation. The **replace-pattern** attribute replaces the given pattern with another pattern either globally or at the indicated hierarchy or object level in the configuration. For example, you can use this feature to find and replace all occurrences of an interface name when a PIC is moved to another slot in the router. The functionality of the attribute is identical to that of the **replace pattern** configuration mode command in the Junos OS CLI.

To replace a pattern, a client application emits the **<rpc>** and **<load-configuration>** tag elements and includes the basic Junos XML tag elements described in *Creating, Modifying, or Deleting Configuration Elements Using the Junos XML Protocol*. At the hierarchy or object level where the pattern should be replaced, the client includes the **replace-pattern** attribute, which specifies the pattern to replace, the **with** attribute, which specifies the replacement pattern, and optionally includes the **upto** attribute, which indicates the number of occurrences to replace. If the **upto** attribute is omitted or set to zero, Junos OS replaces all instances of the pattern within the specified scope. The placement of the attributes within the configuration determines the scope as described in the following sections:

- [Replacing Patterns Globally Within the Configuration on page 108](#)
- [Replacing Patterns Within a Hierarchy Level or Container Object That Has No Identifier on page 109](#)
- [Replacing Patterns for a Configuration Object That Has an Identifier on page 110](#)

Replacing Patterns Globally Within the Configuration

To globally replace a pattern throughout the candidate configuration or open configuration database, include the **replace-pattern** and **with** attributes in the opening **<configuration>** tag.

```
<rpc>
  <load-configuration>
    <configuration replace-pattern="pattern1" with="pattern2" [upto="n"]>
    </configuration>
  </load-configuration>
</rpc>
```

For example, the following RPC replaces all instances of 172.17.1.5 with 172.10.1.1:

```
<rpc>
  <load-configuration>
    <configuration replace-pattern="172.17.1.5" with="172.10.1.1">
    </configuration>
  </load-configuration>
</rpc>
```

After executing the RPC, you can compare the updated candidate configuration to the active configuration to verify the pattern replacement. You must commit the configuration for the changes to take effect.


```

<rpc>
  <get-configuration compare="rollback" rollback="0" format="text">
  </get-configuration>
</rpc>

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
<configuration-information>
<configuration-output>
[edit groups global system ntp]
-   boot-server 172.17.1.5;
+   boot-server 172.10.1.1;
[edit groups global system ntp]
+   server 172.10.1.1;
-   server 172.17.1.5;
</configuration-output>
</configuration-information>
</rpc-reply>

```

Replacing Patterns Within a Hierarchy Level or Container Object That Has No Identifier

To replace a pattern under a specific hierarchy level including all of its children (or a container object that has children but no identifier), a client application includes the **replace-pattern** and **with** attributes in the empty tag that represents the hierarchy level or container object.

```

<rpc>
  <load-configuration>
    <configuration>
      <!-- opening tag for each parent element -->
      <level-or-object replace-pattern="pattern1" with="pattern2" [upto="n"]/>
      <!-- closing tag for each parent element -->
    </configuration>
  </load-configuration>
</rpc>

```

The following RPC replaces instances of fe-0/0/1 with ge-1/0/1 at the **[edit interfaces]** hierarchy level:

```

<rpc>
  <load-configuration>
    <configuration>
      <interfaces replace-pattern="fe-0/0/1" with="ge-1/0/1"/>
    </configuration>
  </load-configuration>
</rpc>

```

After executing the RPC, you can compare the updated candidate configuration to the active configuration to verify the pattern replacement. For example:

```

<rpc>
  <get-configuration compare="rollback" rollback="0" format="text">
  </get-configuration>
</rpc>

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
<configuration-information>
<configuration-output>
[edit interfaces]
-   fe-0/0/1 {

```

```

-         unit 0 {
-             family inet {
-                 address 10.0.1.1/27;
-             }
-         }
-     }
+     ge-1/0/1 {
+         unit 0 {
+             family inet {
+                 address 10.0.1.1/27;
+             }
+         }
+     }
</configuration-output>
</configuration-information>
</rpc-reply>

```

Replacing Patterns for a Configuration Object That Has an Identifier

To replace a pattern for a configuration object that has an identifier, a client application includes the **replace-pattern** and **with** attributes in the opening tag for the object, which then encloses the identifier tag element for that object. In the following example, the identifier tag element is **<name>**:

```

<rpc>
  <load-configuration>
    <configuration>
      <!-- opening tag for each parent element -->
      <container-tag replace-pattern="pattern1" with="pattern2" [upto="n"]>
        <name>identifier</name>
      </container-tag>
      <!-- closing tag for each parent element -->
    </configuration>
  </load-configuration>
</rpc>

```

The following RPC replaces instances of "4.5" with "4.1", but only for the fe-0/0/2 interface under the **[edit interfaces]** hierarchy:

```

<rpc>
  <load-configuration>
    <configuration>
      <interfaces>
        <interface replace-pattern="4.5" with="4.1">
          <name>fe-0/0/2</name>
        </interface>
      </interfaces>
    </configuration>
  </load-configuration>
</rpc>

```

After executing the RPC, you can compare the updated candidate configuration to the active configuration to verify the pattern replacement. For example:

```

<rpc>
  <get-configuration compare="rollback" rollback="0" format="text">
  </get-configuration>
</rpc>

```

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/15.1R1/junos">
<configuration-information>
<configuration-output>
[edit interfaces fe-0/0/2 unit 0 family inet]
+      address 10.0.4.1/30;
-      address 10.0.4.5/30;
</configuration-output>
</configuration-information>
```

- Related Documentation**
- [replace-pattern on page 170](#)
 - *Using Global Replace in a Junos OS Configuration*
 - *Common Regular Expressions to Use with the replace Command*
 - *replace*

Committing the Configuration

- [Verifying the Configuration Syntax Using NETCONF on page 113](#)
- [Committing the Candidate Configuration Using NETCONF on page 114](#)
- [Committing the Candidate Configuration Only After Confirmation Using NETCONF on page 114](#)

Verifying the Configuration Syntax Using NETCONF

In a NETCONF session with a device running Junos OS, during the process of committing the candidate configuration or a private copy, the NETCONF server confirms that the configuration is syntactically correct. If the syntax check fails, the server does not commit the candidate configuration. To avoid the potential complications of such a failure, it often makes sense to confirm the correctness of the candidate configuration before actually committing it.

In a NETCONF session with a device running Junos OS, to verify the syntax of the candidate configuration, a client application includes the `<validate>` and `<source>` tag elements and `<candidate/>` tag in an `<rpc>` tag element:

```
<rpc>
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>
]]>]]>
```

The NETCONF server confirms that the candidate configuration syntax is valid by returning the `<ok/>` tag in the `<rpc-reply>` tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

If the candidate configuration syntax is not valid, the server returns the `<rpc-reply>` element and `<rpc-error>` child element, which explains the reason for the error.

Related Documentation

- [Committing the Candidate Configuration Using NETCONF on page 114](#)

- [Committing the Candidate Configuration Only After Confirmation Using NETCONF on page 114](#)

Committing the Candidate Configuration Using NETCONF

When you commit the candidate configuration on a device running Junos OS, it becomes the active configuration on the routing, switching, or security platform. For more detailed information about commit operations, including a discussion of the interaction among different variants of the operation, see the *CLI User Guide*.

In a NETCONF session with a device running Junos OS, to commit the candidate configuration, a client application encloses the `<commit/>` tag in an `<rpc>` tag element.

```
<rpc>
  <commit/>
</rpc>
]]>]]>
```

We recommend that the client application lock the candidate configuration before modifying it and emit the `<commit/>` tag while the configuration is still locked. This process avoids inadvertently committing changes made by other users or applications. After committing the configuration, the application must unlock it in order for other users and applications to make changes.

The NETCONF server confirms that the commit operation was successful by returning the `<ok/>` tag in the `<rpc-reply>` tag element:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

If the commit operation fails, the server returns the `<rpc-reply>` element and `<rpc-error>` child element, which explains the reason for the failure. The most common causes are semantic or syntactic errors in the candidate configuration.

Related Documentation

- [Committing the Candidate Configuration Only After Confirmation Using NETCONF on page 114](#)
- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)

Committing the Candidate Configuration Only After Confirmation Using NETCONF

When you commit the candidate configuration on a device running Junos OS, it becomes the active configuration on the routing, switching, or security platform. For more detailed information about commit operations, including a discussion of the interaction among different variants of the operation, see the *CLI User Guide*.

When you commit the candidate configuration, you can require an explicit confirmation for the commit to become permanent. The confirmed commit operation is useful for verifying that a configuration change works correctly and does not prevent management

access to the device. If the change prevents access or causes other errors, the automatic rollback to the previous configuration restores access after the rollback deadline passes. If the commit is not confirmed within the specified amount of time, which is 600 seconds (10 minutes) by default, the device automatically retrieves and commits (rolls back to) the previously committed configuration.

In a NETCONF session with a device running Junos OS, to commit the candidate configuration but require an explicit confirmation for the commit to become permanent, a client application encloses the empty `<confirmed/>` tag in the `<commit>` and `<rpc>` tag elements:

```
<rpc>
  <commit>
    <confirmed/>
  </commit>
</rpc>
]]>]]>
```

To specify a different number of seconds for the rollback deadline, the application encloses a positive integer value in the `<confirm-timeout>` tag element:

```
<rpc>
  <commit>
    <confirmed/>
    <confirm-timeout>rollback-delay</confirm-timeout>
  </commit>
</rpc>
]]>]]>
```

In either case, the NETCONF server confirms that it committed the candidate configuration temporarily by returning the `<ok/>` tag in the `<rpc-reply>`.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>
```

If the NETCONF server cannot commit the candidate configuration, the `<rpc-reply>` element instead encloses an `<rpc-error>` element explaining the reason for the failure. The most common causes are semantic or syntactic errors in the candidate configuration.

To delay the rollback to a time later than the current rollback deadline, the client application emits the `<confirmed/>` tag in a `<commit>` tag element again before the deadline passes. Optionally, it includes the `<confirm-timeout>` element to specify how long to delay the next rollback; omit that tag element to delay the rollback by the default of 600 seconds (10 minutes). The client application can delay the rollback indefinitely by emitting the `<confirmed/>` tag repeatedly in this way.

To commit the configuration permanently, the client application emits the `<commit/>` tag enclosed in an `<rpc>` tag element before the rollback deadline passes. The rollback is canceled and the candidate configuration is committed immediately, as described in [“Committing the Candidate Configuration Using NETCONF” on page 114](#). If the candidate configuration is still the same as the temporarily committed configuration, this effectively recommits the temporarily committed configuration.

If another application uses the `<kill-session/>` tag element to terminate this application's session while a confirmed commit is pending (this application has committed changes but not yet confirmed them), the NETCONF server that is servicing this session restores the configuration to its state before the confirmed commit instruction was issued. For more information about session termination, see [“Terminating a NETCONF Session” on page 74](#).

The following example shows how to commit the candidate configuration with a rollback deadline of 300 seconds.

Client Application	<pre><rpc> <commit> <confirmed/> <confirm-timeout>300</confirm-timeout> </commit> </rpc>]]>]]></pre>
NETCONF Server	<pre><rpc-reply xmlns="URN" xmlns:junos="URL"> <ok/> </rpc-reply>]]>]]></pre>

Related Documentation

- [Committing the Candidate Configuration Using NETCONF on page 114](#)

CHAPTER 12

Configuration Statements

- [connection-limit on page 118](#)
- [netconf on page 119](#)
- [port \(NETCONF\) on page 120](#)
- [rate-limit on page 121](#)
- [rfc-compliant \(NETCONF\) on page 122](#)
- [ssh \(NETCONF\) on page 123](#)
- [traceoptions \(NETCONF\) on page 124](#)

connection-limit

Syntax	<code>connection-limit <i>limit</i>;</code>
Hierarchy Level	[edit system services finger], [edit system services ftp], [edit system services netconf ssh], [edit system services ssh], [edit system services telnet], [edit system services xnm-clear-text], [edit system services xnm-ssl]
Release Information	Statement introduced before Junos OS Release 7.4. Statement introduced in Junos OS Release 9.0 for EX Series switches. Statement introduced in Junos OS Release 14.1X53-D20 for OCX Series switches. Statement introduced in Junos OS Release 11.1 for the QFX Series.
Description	Configure the maximum number of connections sessions for each type of system services (finger, ftp, ssh, telnet, xnm-clear-text, or xnm-ssl) per protocol (either IPv6 or IPv4).
Options	limit —(Optional) Maximum number of established connections per protocol (either IPv6 or IPv4). Range: 1 through 250 Default: 75



NOTE: The actual number of maximum connections depends on the availability of system resources, and might be fewer than the configured `connection-limit` value if the system resources are limited.

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

Related Documentation	<ul style="list-style-type: none">• <i>Configuring clear-text or SSL Service for Junos XML Protocol Client Applications</i>• <i>Configuring DTCP-over-SSH Service for the Flow-Tap Application</i>• <i>Configuring Finger Service for Remote Access to the Router</i>• <i>Configuring FTP Service for Remote Access to the Router or Switch</i>• <i>Configuring SSH Service for Remote Access to the Router or Switch</i>• <i>Configuring Telnet Service for Remote Access to a Router or Switch</i>
------------------------------	---

netconf

Syntax	<pre> netconf { rfc-compliant; ssh { connection-limit <i>limit</i>; port <i>port</i>; rate-limit <i>limit</i>; } traceoptions { file <filename> <files <i>number</i>> <match <i>regular-expression</i>> <size <i>size</i>> <world-readable no-world-readable>; flag <i>flag</i>; no-remote-trace; on-demand; } } </pre>
Hierarchy Level	[edit system services]
Release Information	Statement introduced in Junos OS Release 7.5.
Description	<p>Configure the NETCONF XML management protocol.</p> <p>The remaining statements are explained separately.</p>
Default	If you do not include the netconf statement, NETCONF connections are not permitted.
Required Privilege Level	<p>system—To view this statement in the configuration.</p> <p>system-control—To add this statement to the configuration.</p>
Related Documentation	<ul style="list-style-type: none"> • connection-limit on page 118 • port (NETCONF) on page 120 • rate-limit on page 121 • ssh (NETCONF) on page 123 • traceoptions (NETCONF) on page 124

port (NETCONF)

Syntax	<code>port <i>port-number</i>;</code>
Hierarchy Level	[edit system services netconf]
Release Information	Statement introduced in Junos OS Release 10.0.
Description	Configure the TCP port used for NETCONF-over-SSH connections.



NOTE:

- The configured port accepts only NETCONF-over-SSH connections. Regular SSH session requests for this port are rejected.
 - The default SSH port (22) continues to accept NETCONF sessions even with a configured NETCONF server port. To disable the SSH port from accepting NETCONF sessions, you can specify this in the login event script.
 - We do not recommend configuring the default ports for FTP (21) and Telnet (23) services for configuring NETCONF-over-SSH connections.
-

Options	port <i>port-number</i> —Port number on which to enable incoming NETCONF connections over SSH. Default: 830 (as specified in RFC 4742, <i>Using the NETCONF Configuration Protocol over Secure Shell (SSH)</i>) Range: 1 through 65535
Required Privilege Level	system—To view this statement in the configuration. system-control—To add this statement to the configuration.
Related Documentation	<ul style="list-style-type: none">• <i>NETCONF XML Management Protocol Guide</i>• <i>Configuring NETCONF-Over-SSH Connections on a Specified TCP Port</i>

rate-limit

Syntax	<code>rate-limit <i>limit</i>;</code>
Hierarchy Level	<code>[edit system services finger],</code> <code>[edit system services ftp],</code> <code>[edit system services netconf ssh],</code> <code>[edit system services ssh],</code> <code>[edit system services telnet],</code> <code>[edit system services xnm-clear-text],</code> <code>[edit system services xnm-ssl]</code>
Release Information	<p>Statement introduced before Junos OS Release 7.4.</p> <p>Statement introduced in Junos OS Release 9.0 for EX Series switches.</p> <p>Statement introduced in Junos OS Release 14.1X53-D20 for OCX Series switches.</p> <p>Statement introduced in Junos OS Release 11.1 for the QFX Series.</p>
Description	Configure the maximum number of connections attempts per protocol (either IPv6 or IPv4) on an access service.
Default	150 connections
Options	<p>rate-limit <i>limit</i>—(Optional) Maximum number of connection attempts allowed per minute, per IP protocol (either IPv4 or IPv6).</p> <p>Range: 1 through 250</p> <p>Default: 150</p>
Required Privilege Level	<p>system—To view this statement in the configuration.</p> <p>system-control—To add this statement to the configuration.</p>
Related Documentation	<ul style="list-style-type: none"> <i>Configuring clear-text or SSL Service for Junos XML Protocol Client Applications</i>

rfc-compliant (NETCONF)

Syntax	rfc-compliant;
Hierarchy Level	[edit system services netconf]
Release Information	Statement introduced in Junos OS Release 15.1.
Description	<p>Require that the NETCONF server enforce certain behaviors that are compliant with RFC 4741, <i>NETCONF Configuration Protocol</i>, during NETCONF sessions.</p> <p>When you configure the rfc-compliant statement:</p> <ul style="list-style-type: none">• The NETCONF server explicitly declares the NETCONF namespace in its replies and qualifies all NETCONF tags with the nc prefix• <get> and <get-config> operations that return no configuration data do not include an empty <configuration> element in RPC replies
Default	If you do not include the rfc-compliant statement, the NETCONF server sets the default namespace to the NETCONF namespace in RPC replies, and <get> and <get-config> operations that return no configuration data include an empty <configuration> element in RPC replies.
Required Privilege Level	system—To view this statement in the configuration. system-control—To add this statement to the configuration.
Related Documentation	<ul style="list-style-type: none">• Configuring RFC-Compliant NETCONF Sessions on page 79

ssh (NETCONF)

Syntax	<pre>ssh { <i>connection-limit limit</i>; <i>port port-number</i>; <i>rate-limit limit</i>; }</pre>
Hierarchy Level	[edit system services netconf]
Description	Enable access to the NETCONF SSH subsystem using the default port number 830, as specified by RFC 4742.
Options	The statements are explained separately.
Required Privilege Level	system—To view this statement in the configuration. system-control—To add this statement to the configuration.
Related Documentation	<ul style="list-style-type: none">• connection-limit on page 118• netconf on page 119• port (NETCONF) on page 120• rate-limit on page 121

traceoptions (NETCONF)

Syntax	<pre>traceoptions { file <filename> <files number> <match regular-expression> <size size> <world-readable no-world-readable>; flag flag; no-remote-trace; on-demand; }</pre>
Hierarchy Level	[edit system services netconf]
Release Information	Statement introduced in Junos OS Release 12.2.
Description	Define tracing operations for NETCONF sessions.
Default	If you do not include this statement, NETCONF-specific tracing operations are not performed.
Options	<p>file filename—Name of the file to receive the output of the tracing operation. All files are placed in the <code>/var/log</code> directory.</p> <p>Default: <code>/var/log/netconf</code></p> <p>files number—(Optional) Maximum number of trace files. When a trace file named trace-file reaches its maximum size, it is renamed and compressed to trace-file.0.gz. When trace-file again reaches its maximum size, trace-file.0.gz is renamed trace-file.1.gz, and trace-file is renamed and compressed to trace-file.0.gz. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.</p> <p>If you specify a maximum number of files, you also must specify a maximum file size with the size option and a filename.</p> <p>Range: 2 through 1000 files</p> <p>Default: 10 files</p> <p>flag flag—Tracing operation to perform. To specify more than one tracing operation, include multiple flag statements. You can include the following flags:</p> <ul style="list-style-type: none">• all—Log all incoming and outgoing data from NETCONF sessions.• incoming—Log all incoming data from NETCONF sessions.• outgoing—Log all outgoing data from NETCONF sessions. <p>match regular-expression—(Optional) Refine the output to include only those lines that match the regular expression.</p> <p>no-remote-trace—(Optional) Disable remote tracing.</p> <p>no-world-readable—(Optional) Disable unrestricted file access, which restricts file access to the owner. This is the default.</p>

on-demand—(Optional) Enable on-demand tracing, which requires that you start and stop tracing operations from within the NETCONF session. If configured, tracing operations are performed for a NETCONF session only when requested through the `<request-netconf-trace>` operation.

Within a NETCONF session, issue the

`<request-netconf-trace><start/></request-netconf-trace>` RPC to start tracing operations for that session, and issue the

`<request-netconf-trace><stop/></request-netconf-trace>` RPC to stop tracing operations for that session.

size *size*—(Optional) Maximum size of each trace file in bytes, kilobytes (KB), megabytes (MB), or gigabytes (GB). If you don't specify a unit, the default is bytes. If you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option and a filename.

Syntax: *size* to specify bytes, *sizek* to specify KB, *sizem* to specify MB, or *sizeg* to specify GB

Range: 10240 through 1073741824 bytes

Default: 128 KB

world-readable—(Optional) Enable unrestricted file access.

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

Related Documentation	<ul style="list-style-type: none"> • NETCONF Tracing Operations Overview on page 45 • Example: Configuring NETCONF Tracing Operations on page 46 • netconf on page 119
------------------------------	---

NETCONF Protocol Operations

<close-session/>

Usage	<pre><rpc> <close-session/> </rpc>]]>]]></pre>
Description	Request that the NETCONF server end the current session.
Related Documentation	<ul style="list-style-type: none"> • Ending a NETCONF Session and Closing the Connection on page 75 •]]>]]> on page 137 • <rpc> on page 140

<commit>

Usage	<pre><rpc> <commit/> </rpc>]]>]]> <rpc> <commit> <confirmed/> <confirm-timeout>rollback-delay</confirm-timeout> </commit> </rpc>]]>]]></pre>
Description	<p>Request that the NETCONF server perform one of the variants of the commit operation on the candidate configuration:</p> <ul style="list-style-type: none"> • To commit the configuration immediately, making it the active configuration on the device, emit the empty <commit/> tag. • To commit the candidate configuration but require an explicit confirmation for the commit to become permanent, enclose the <confirmed/> tag in the <commit> tag element.

By default, the NETCONF server rolls back to the previous running configuration after 600 seconds (10 minutes); to set a different rollback delay, also emit the optional `<confirm-timeout>` tag element. To delay the rollback again (past the original rollback deadline), emit the `<confirmed/>` tag (enclosed in the `<commit>` tag element) again before the deadline passes. Include the `<confirm-timeout>` tag element to specify how long to delay the next rollback, or omit that tag element to use the default of 600 seconds (10 minutes). The rollback can be delayed repeatedly in this way.

To commit the configuration immediately and permanently after emitting the `<confirmed/>` tag, emit the empty `<commit/>` tag before the rollback deadline passes. The NETCONF server commits the candidate configuration and cancels the rollback. If the candidate configuration is still the same as the running configuration, the effect is the same as recommitting the current running configuration.

Contents `<confirmed>`—Requests a temporary commit of the candidate configuration. The device reverts to the previous active configuration after a specified time.

`<confirm-timeout>`—Specifies the number of seconds before the device reverts to the previously active configuration. If this tag element is omitted, the default is 600 seconds (10 minutes).

Related Documentation

- [Committing the Candidate Configuration Using NETCONF on page 114](#)
- [Committing the Candidate Configuration Only After Confirmation Using NETCONF on page 114](#)

`<copy-config>`

Usage

```
<rpc>
  <copy-config>
    <target>
      <candidate/>
    </target>
    <source>
      <url format="(xml | text)">
        <!-- location specifier for file containing the new configuration -->
      </url>
    </source>
  </copy-config>
</rpc>
]]>]]>
```

Description Replace the existing candidate configuration with configuration data contained in a file.

Contents `<source>`—Encloses the `<url>` tag element, which specifies the source of the configuration data.

`<url>`—Names the file that contains the new configuration data to substitute for the existing candidate configuration. When the configuration data is formatted as Junos XML tag elements, set the `<url>` **format** attribute to **xml** or omit the attribute. When

the configuration data is formatted as CLI configuration statements, set the `<url>` format attribute to `text`. For more information, see [“Uploading and Formatting Configuration Data in a NETCONF Session” on page 85](#).

The `<target>` tag element and its contents are explained separately.

- Related Documentation**
- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
 - [<target> on page 141](#)

`<delete-config>`

Usage

```
<rpc>
  <delete-config>
    <target>
      <candidate/>
    </target>
  <delete-config>
</rpc>
]]>]]>
```

Description Delete the existing candidate configuration.

Contents The `<target>` tag element and its contents are explained separately.

- Related Documentation**
- [Deleting the Candidate Configuration Using NETCONF on page 96](#)
 - [Deleting Configuration Elements Using NETCONF on page 101](#)
 - [<target> on page 141](#)

`<discard-changes/>`

Usage

```
<rpc>
  <discard-changes/>
</rpc>
]]>]]>
```

Description Discard changes made to the candidate configuration and make its contents match the contents of the current running (active) configuration. This operation is equivalent to the Junos OS CLI configuration mode **rollback 0** command.

- Related Documentation**
- [Rolling Back the Candidate Configuration Using NETCONF on page 95](#)

`<edit-config>`

Usage

```
<rpc>
  <edit-config>
```

```
<target>
  <candidate/>
</target>

<!-- EITHER -->

<config>
  <configuration>
    <!-- tag elements representing the data to incorporate -->
  </configuration>
</config>

<!-- OR -->

<config-text>
  <configuration-text>
    <!-- configuration data in text format -->
  </configuration-text>
</config-text>

<!-- OR -->

<url format="(xml | text)">
  <!-- location specifier for file containing data -->
</url>

<default-operation>(merge | none | replace)</default-operation>
<error-option>(ignore-error | stop-on-error)</error-option>
<test-option>(set | test-then-set)</test-option>
<edit-config>
</rpc>
]]>]]>
```

Description Request that the NETCONF server incorporate configuration data into the candidate configuration. Provide the data in one of three ways:

- Include the **<config>** tag element to provide a data stream of Junos XML configuration tag elements to incorporate. The tag elements are enclosed in the **<configuration>** tag element.
- Include the **<config-text>** tag element to provide a data stream of CLI configuration statements to incorporate. The configuration statements are enclosed in the **<configuration-text>** tag element.
- Include the **<url>** tag element to specify the location of a file that contains the Junos XML configuration tag elements to incorporate.

Contents **<config>**—Encloses the **<configuration>** tag element.

<configuration>—Encloses the configuration data written in Junos XML. This configuration data will be incorporated into the candidate configuration and provided as a data stream. For information about the syntax for representing the elements to create, delete, or modify, see [“Mapping Configuration Statements to Junos XML Tag Elements” on page 15](#).

`<config-text>`—Encloses the `<configuration-text>` tag element.

`<configuration-text>`—Encloses the configuration data written in CLI configuration statements. This configuration data will be incorporated into the candidate configuration and provided as a data stream.

`<default-operation>`—(Optional) Specifies how to incorporate the new configuration data into the candidate configuration, particularly when there are conflicting statements. The following are acceptable values:

- `merge`—Combines the new configuration data with the candidate configuration according to the rules defined in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). This is the default mode if the `<default-operation>` tag element is omitted. It applies to all elements in the new data that do not have the `operation` attribute in their opening container tag to specify a different mode.
- `none`—Retains each configuration element in the existing candidate configuration unless the new data includes a corresponding element that has the `operation` attribute in its opening container tag to specify an incorporation mode. This mode prevents the NETCONF server from creating parent hierarchy levels for an element that is being deleted. For more information, see [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#).
- `replace`—Discards the existing candidate configuration and replaces it with the new data. For more information, see [“Replacing the Candidate Configuration Using NETCONF” on page 94](#).

`<error-option>`—(Optional) Specifies how the NETCONF server handles errors encountered while it incorporates the configuration data. The following are acceptable values:

- `ignore-error`—Specifies that the NETCONF server continue to incorporate the new configuration data even if it encounters an error.
- `stop-on-error`—Specifies that the NETCONF server stop incorporating the new configuration data when it encounters an error. This is the default behavior if the `<error-option>` tag element is omitted.

`<test-option>`—(Optional) Specifies whether the NETCONF server validates the configuration data before incorporating it into the candidate configuration. The acceptable values defined in the NETCONF specification are `set` (no validation) and the default `test-then-set` (do not incorporate data if validation fails).

Regardless of the value provided, the NETCONF server for the Junos OS performs a basic syntax check on the configuration data in the `<edit-config>` tag element. It performs a complete syntactic and semantic validation in response to the `<validate>` and `<commit>` tag elements, but not for the `<edit-config>` tag element.

`<url>`—Specifies the full pathname of the file that contains the configuration data to load. When the configuration data is formatted as Junos XML tag elements, set the `<url>` `format` attribute to `xml` or omit the attribute. When the configuration data is formatted as CLI configuration statements, you set the `<url>` `format` attribute to `text`. For more information, see [“Uploading and Formatting Configuration Data in a NETCONF Session” on page 85](#).

The **<target>** tag element and its contents are explained separately.

- Related Documentation**
- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
 - [Editing the Candidate Configuration Using NETCONF on page 83](#)
 - [Replacing the Candidate Configuration Using NETCONF on page 94](#)
 - [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)
 - [Uploading and Formatting Configuration Data in a NETCONF Session on page 85](#)
 - [<target> on page 141](#)

<get>

Usage

```
<rpc>
  <get [format="(set | text | xml)"]>
    <filter type="subtree">
      <configuration>
        <!-- tag elements representing the configuration elements to return -->
      </configuration>
    </filter>
  </get>
</rpc>
]]>]]>
```

Description Request the committed configuration and device state information from the NETCONF server. To display one or more sections of the configuration hierarchy (hierarchy levels or configuration objects), enclose the appropriate child tag elements in the **<filter>** element.

Attributes format—(Optional) Specifies the return format for the configuration data. If you omit this attribute, the server returns the configuration data formatted as Junos XML elements.

Contents <filter>—(Optional) Encloses the **<configuration>** tag element. The optional **type** attribute indicates the kind of syntax used to represent the requested configuration elements; the only acceptable value is **subtree**.

To specify the configuration elements to return, include within the **<filter>** tag element the Junos XML tag elements that represent all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to each element to display. For information about the configuration elements available in the current version of the Junos OS, see *Junos XML API Configuration Developer Reference*.

- Related Documentation**
- [Requesting the Committed Configuration and Device State Using NETCONF on page 183](#)

<get-config>

Usage

```

<rpc>
  <get-config>
    <source>
      <( candidate | running )/>
    </source>
  </get-config>

  <get-config>
    <source>
      <( candidate | running )/>
    </source>
    <filter type="subtree">
      <configuration>
        <!-- tag elements for each configuration element to return -->
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>

```

Description Request configuration data from the NETCONF server. The child tag elements **<source>** and **<filter>** specify the source and scope of data to display:

- To display the entire active configuration, enclose the **<source>** tag element and **<running/>** tag in the **<get-config>** tag element.
- To display the entire candidate configuration, enclose the **<source>** tag element and **<candidate/>** tag in the **<get-config>** tag element.
- To display one or more sections of the configuration hierarchy (hierarchy levels or configuration objects), enclose the appropriate child tag elements in the **<source>** and **<filter>** tag elements.

Contents **<candidate/>**—Specifies the candidate configuration.

<configuration>—Encloses tag elements that specify which configuration elements to return.

<filter>—Encloses the **<configuration>** tag element. The mandatory **type** attribute indicates the kind of syntax used to represent the requested configuration elements; the only acceptable value is **subtree**.

To specify the configuration elements to return, include within the **<filter>** tag element the Junos XML tag elements that represent all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to each element to display. For information about the configuration elements available in the current version of the Junos OS, see the *Junos XML API Configuration Developer Reference*.

<running/>—Represents the active (mostly recently committed) configuration.

`<source>`—Encloses the tag that specifies the source of the configuration data. To specify the candidate configuration, include the `<candidate/>` tag. To specify the active configuration, include the `<running/>` tag.

Usage Guidelines See [“Requesting Configuration Data Using NETCONF” on page 184](#).

Related Documentation

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- [<data> on page 137](#)

`<kill-session>`

Usage

```
<rpc>
  <kill-session>
    <session-id>PID</session-id>
  </kill-session>
</rpc>
]]>]]>
```

Description Request that the NETCONF server terminate another CLI or NETCONF session. The usual reason to emit this tag is that the user or application for the other session holds a lock on the candidate configuration, preventing the client application from locking the configuration itself.

The client application must have the Junos **maintenance** permission to perform this operation.

Contents `<session-id>`—The process identifier (PID) of the entity conducting the session to terminate. The PID is reported in the `<rpc-error>` tag element that the NETCONF server generates when it cannot lock a configuration as requested.

Related Documentation

- [Terminating a NETCONF Session on page 74](#)
- [<lock> on page 134](#)
- [<rpc-error> on page 140](#)

`<lock>`

Usage

```
<rpc>
  <lock>
    <target>
      <candidate/>
    </target>
  </lock>
</rpc>
]]>]]>
```

Description Request that the NETCONF server lock the candidate configuration, enabling the client application both to read and change it, but preventing any other users or applications from changing it. The client application must emit the `<unlock/>` tag to unlock the configuration.

If the NETCONF session ends or the application emits the `<unlock>` tag element before the candidate configuration is committed, all changes made to the candidate are discarded.

Contents The `<target>` tag element and its contents are explained separately.

Related Documentation

- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)
- [<rpc> on page 140](#)
- [<target> on page 141](#)
- [<unlock> on page 135](#)

<unlock>

Usage

```
<rpc>
  <unlock>
    <target>
      <candidate/>
    </target>
  </unlock>
</rpc>
]]>]]>
```

Description Request that the NETCONF server unlock and close the candidate configuration, which the client application previously locked by emitting the `<lock>` tag element. Until the application emits this tag element, other users or applications can read the configuration but cannot change it.

Contents The `<target>` tag element and its contents are explained separately.

Related Documentation

- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)
- [<lock> on page 134](#)
- [<target> on page 141](#)

<validate>

Usage

```
<rpc>
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
```

```
</rpc>  
]]>]]>
```

Description Check that the candidate configuration is syntactically valid.

Contents <source>—Encloses the tag that specifies the configuration to validate.
<candidate/>—Specifies the candidate configuration.

Related Documentation

- [Verifying the Configuration Syntax Using NETCONF on page 113](#)

NETCONF Request and Response Tags

]]>]]>

Usage

```

<hello>
  <!-- child tag elements included by client application or NETCONF server -->
</hello>
]]>]]>

<rpc [attributes]>
  <!-- tag elements in a request from a client application -->
</rpc>
]]>]]>

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <!-- tag elements in the response from the NETCONF server -->
</rpc-reply>
]]>]]>

```

Description Signal the end of each XML document sent by the NETCONF server and client applications. Client applications send the sequence after its closing `</hello>` tag and each closing `</rpc>` tag. The NETCONF server sends the sequence after its closing `</hello>` tag and each closing `</rpc-reply>` tag.

Use of this signal is required by RFC 4742, *Using the NETCONF Configuration Protocol over Secure SHell (SSH)*, available at <http://www.ietf.org/rfc/rfc4742.txt>.

- Related Documentation**
- [Generating Well-Formed XML Documents on page 24](#)
 - [<hello> on page 138](#)
 - [<rpc> on page 140](#)
 - [<rpc-reply> on page 141](#)

<data>

Usage

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration>
      <!-- Junos XML tag elements for the configuration data -->
    </configuration>
  </data>
</rpc-reply>

```

```
</data>
</rpc-reply>
]]>]]>
```

Description	Enclose configuration data and device information returned by the NETCONF server in response to a <get> request and configuration data returned by the NETCONF server in response <get-config> request.
Contents	<configuration> —Encloses configuration tag elements. It is the top-level tag element in the Junos XML API, equivalent to the [edit] hierarchy level in the Junos OS CLI. For information about Junos configuration elements, see the <i>Junos XML API Configuration Developer Reference</i> .
Usage Guidelines	See “ Requesting Configuration Data Using NETCONF ” on page 184.
Related Documentation	<ul style="list-style-type: none">• <configuration> in the <i>Junos XML API Configuration Developer Reference</i>• <get-config> on page 133• <rpc-reply> on page 141

<error-info>

Usage	<pre><rpc-reply xmlns="URN" xmlns:junos="URL"> <rpc-error> <error-info> <bad-element>command-or-statement</bad-element> </error-info> </rpc-error> </rpc-reply>]]>]]></pre>
Description	Provide additional information about the event or condition that causes the NETCONF server to report an error or warning in the <rpc-error> tag element.
Contents	<bad-element> —Identifies the command or configuration statement that was being processed when the error or warning occurred. For a configuration statement, the <error-path> tag element enclosed in the <rpc-error> tag element specifies the statement's parent hierarchy level.
Related Documentation	<ul style="list-style-type: none">• Handling an Error or Warning in a NETCONF Session on page 70• <rpc-error> on page 140• <rpc-reply> on page 141

<hello>

Usage	<pre><!-- emitted by a client application --></pre>
--------------	---

```

<hello>
  <capabilities>
    <capability>URI</capability>
  </capabilities>
</hello>
]]>]]>

<!-- emitted by the NETCONF server -->
<hello>
  <capabilities>
    <capability>URI</capability>
  </capabilities>
  <session-id>session-identifier</session-id>
</hello>
]]>]]>

```

Description Specify which operations, or *capabilities*, the emitter supports from among those defined in the NETCONF specification. The client application must emit the **<hello>** tag element before any other tag element during the NETCONF session, and must not emit it more than once.

Contents **<capabilities>**—Encloses one or more **<capability>** tags, which together specify the set of supported NETCONF operations.

<capability>—Specifies the uniform resource identifier (URI) of a capability defined in the NETCONF specification or by a vendor. Each capability from the NETCONF specification is represented by a uniform resource name (URN). Capabilities defined by vendors are represented by URNs or URLs.

<session-id>—(Generated by NETCONF server only) Specifies the UNIX process ID (PID) of the NETCONF server for the session.

Related Documentation

- [Exchanging <hello> Tag Elements on page 61](#)

<ok/>

Usage

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
</rpc-reply>
]]>]]>

```

Description Indicate that the NETCONF server successfully performed a requested operation that changes the state or contents of the device configuration.

Related Documentation

- [Configuration Change Responses on page 69](#)
- [<rpc-reply> on page 141](#)

<rpc>

Usage	<pre><rpc [attributes]>] <!-- tag elements in a request from a client application --> </rpc>]]>]]></pre>
Description	Enclose all tag elements in a request generated by a client application.
Attributes	(Optional) One or more attributes of the form <i>attribute-name</i> ="value". This feature can be used to associate requests and responses if the value assigned to an attribute by the client application is unique in each opening <rpc> tag. The NETCONF server echoes the attribute unchanged in its opening <rpc-reply> tag, making it simple to map the response to the initiating request. The NETCONF specification assigns the name message-id to this attribute.
Related Documentation	<ul style="list-style-type: none">• Sending a Request to the NETCONF Server on page 65• <rpc-reply> on page 141

<rpc-error>

Usage	<pre><rpc-reply xmlns="URN" xmlns:junos="URL"> <rpc-error> <error-severity>error-severity</error-severity> <error-path>error-path</error-path> <error-message>error-message</error-message> <error-info>...</error-info> </rpc-error> </rpc-reply>]]>]]></pre>
Description	Indicate that the NETCONF server has experienced an error while processing the client application's request. If the server has already emitted the response tag element for the current request, the information enclosed in that response tag element might be incomplete. The client application must include code that discards or retains the information, as appropriate. The child tag elements described in the Contents section detail the nature of the error. The NETCONF server does not necessarily emit all child tag elements; it omits tag elements that are not relevant to the current request.
Contents	<p><error-message>—Describes the error or warning in a natural-language text string.</p> <p><error-path>—Specifies the path to the Junos configuration hierarchy level at which the error or warning occurred, in the form of the CLI configuration mode banner.</p> <p><error-severity>—Indicates the severity of the event that caused the NETCONF server to return the <rpc-error> tag element. The two possible values are error and warning.</p> <p>The <error-info> tag element is described separately.</p>

- Related Documentation**
- [Handling an Error or Warning in a NETCONF Session on page 70](#)
 - [<error-info> on page 138](#)
 - [<rpc-reply> on page 141](#)

<rpc-reply>

- Usage** `<rpc-reply xmlns="URN" xmlns:junos="URL">
 <!-- tag elements in a reply from the NETCONF server-->
 </rpc-reply>
]]>]]>`
- Description** Enclose all tag elements in a reply from the NETCONF server. The immediate child tag element is usually one of the following:
- The Junos XML tag element that encloses the data requested by a client application with a Junos XML operational request tag element; for example, the `<interface-information>` tag element in response to the `<get-interface-information>` tag element
 - The `<data>` tag element, to enclose the data requested by a client application with the `<get-config>` tag element
 - The `<ok/>` tag, to confirm that the NETCONF server successfully performed an operation that changes the state or contents of a configuration (such as a lock, change, or commit operation)
 - The `<output>` tag element, if the Junos XML API does not define a specific tag element for requested operational information
 - The `<rpc-error>` tag element, if the requested operation generated an error or warning
- Attributes** xmlns—Names the default XML namespace for the enclosed tag elements.
- Related Documentation**
- [Parsing the NETCONF Server Response on page 67](#)
 - [<data> on page 137](#)
 - [<ok/> on page 139](#)
 - `<output>` in the *Junos XML API Operational Developer Reference*
 - [<rpc> on page 140](#)
 - [<rpc-error> on page 140](#)

<target>

- Usage** `<rpc>
 <(copy-config | delete-config | edit-config | lock | unlock)>
 <target>
 <candidate/>`

```
    </target>
  </( copy-config | delete-config | edit-config | lock | unlock )>
</rpc>
]]>]]>
```

Description Specify the configuration on which to perform an operation.

Contents <candidate/>—Specifies the candidate configuration as the configuration on which to perform the operation. This is the only acceptable value for the Junos OS.

Related Documentation

- [Deleting the Candidate Configuration Using NETCONF on page 96](#)
- [Editing the Candidate Configuration Using NETCONF on page 83](#)
- [Locking and Unlocking the Candidate Configuration Using NETCONF on page 72](#)
- [Replacing the Candidate Configuration Using NETCONF on page 94](#)
- [<copy-config> on page 128](#)
- [<delete-config> on page 129](#)
- [<edit-config> on page 129](#)
- [<lock> on page 134](#)
- [<unlock> on page 135](#)

Junos XML Protocol Elements Supported in NETCONF Sessions

<abort/>

Usage	<pre><rpc> <!-- child tag elements --> </rpc> <abort/></pre>
Release Information	<p>This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.</p>
Description	<p>Direct the NETCONF or Junos XML protocol server to stop processing the request that is currently outstanding. The server responds by returning the <abort-acknowledgment/> tag, but might already have sent tagged data in response to the request. The client application must discard those tag elements.</p>
Related Documentation	<ul style="list-style-type: none"> • <i>Halting a Request in Junos XML Protocol Sessions</i> • <abort-acknowledgement/> on page 143

<abort-acknowledgement/>

Usage	<pre><rpc-reply xmlns:junos="URL"> <any-child-of-rpc-reply> <abort-acknowledgement/> </any-child-of-rpc-reply> </rpc-reply></pre>
Release Information	<p>This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.</p>

Description Indicate that the NETCONF or Junos XML protocol server has received the `<abort/>` tag and has stopped processing the current request. If the client application receives any tag elements related to the request between sending the `<abort/>` tag and receiving this tag, it must discard them.

Related Documentation

- `<xnm:error>`

`<checksum-information>`

Usage

```
<rpc-reply>
  <checksum-information>
    <file-checksum>
      <computation-method>MD5</computation-method>
      <input-file>
        <!-- name and path of file-->
      </input-file>
    </file-checksum>
  </checksum-information>
</rpc-reply>
```

Release Information This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Enclose tag elements that include the file to check, the checksum algorithm used, and the checksum output.

Contents

- `<checksum>`—Resulting value from the checksum computation.
- `<computation-method>`—Checksum algorithm used. Currently, all checksum computations use the MD5 algorithm; thus, the only possible value is MD5.
- `<file-checksum>`—Wrapper that holds the resulting `<input-file>`, `<computation-method>`, and `<checksum>` attributes for a particular checksum computation.
- `<input-file>`—Name and path of the file that the checksum algorithm was run against.

Related Documentation

- [<get-checksum-information> on page 152](#)

`<close-configuration/>`

Usage

```
<rpc>
  <close-configuration/>
</rpc>
```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF

sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Discard a candidate configuration and any changes to it.

This tag element is normally used only to discard a private copy of the candidate configuration without committing it. The application must have previously emitted the `<open-configuration>` tag element. Closing the NETCONF or Junos XML protocol session (by emitting the `<request-end-session/>` tag, for example) has the same effect as emitting this tag element.

- Related Documentation**
- *Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol*
 - [<open-configuration> on page 161](#)
 - [<request-end-session/> on page 162](#)

`<commit-configuration>`

Usage

```

<rpc>
  <commit-configuration/>

  <commit-configuration>
    <check/>
  </commit-configuration>

  <commit-configuration>
    <log>log-message</log>
  </commit-configuration>

  <commit-configuration>
    <at-time>time-specification</at-time>
    <log>log-message</log>
  </commit-configuration>

  <commit-configuration>
    <confirmed/>
    <confirm-timeout>rollback-delay</confirm-timeout>
    <log>log-message</log>
  </commit-configuration>

  <commit-configuration>
    <synchronize/>
    <log>log-message</log>
  </commit-configuration>

  <commit-configuration>
    <synchronize/>
    <at-time>time-specification</at-time>
    <log>log-message</log>
  </commit-configuration>

```

```
<commit-configuration>
  <synchronize/>
  <check/>
  <log>log-message</log>
</commit-configuration>

<commit-configuration>
  <synchronize/>
  <confirmed/>
  <confirm-timeout>rollback-delay</confirm-timeout>
  <log>log-message</log>
</commit-configuration>

<commit-configuration>
  <synchronize/>
  <force-synchronize/>
</commit-configuration>
</rpc>
```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Request that the NETCONF or Junos XML protocol server perform one of the variants of the commit operation on either the regular candidate configuration or a private copy of the candidate configuration (if the application emitted the `<open-configuration><private/></open-configuration>` tag sequence before making changes).

Some restrictions apply to the commit operation for a private copy. For example, the commit operation fails if the regular candidate configuration is locked by another user or application, or if it includes uncommitted changes made since the private copy was created. For more information, see the *CLI User Guide*.

Enclose the appropriate tag in the `<commit-configuration>` tag element to specify the type of commit operation:

- To commit the configuration immediately, making it the active configuration on the device, emit the empty `<commit-configuration/>` tag.
- To verify the syntactic correctness of the configuration without actually committing it, enclose the `<check/>` tag in the `<commit-configuration>` tag element.
- To record a message in the `/var/log/commits` file when the associated commit operation succeeds, define the log message string in the `<log>` tag element and enclose the tag element in the `<commit-configuration>` tag element. The `<log>` tag element can be combined with any other tag element. When the `<log>` tag element is emitted alone, the associated commit operation begins immediately.

- To commit the candidate configuration but roll back to the previous configuration after a short time, enclose the `<confirmed/>` tag in the `<commit-configuration>` tag element.

By default, the rollback occurs after 10 minutes; to set a different rollback delay, also emit the optional `<confirm-timeout>` tag element. To delay the rollback again (past the original rollback deadline), emit the `<confirmed/>` tag (enclosed in the `<commit-configuration>` tag element) before the deadline passes. Include the `<confirm-timeout>` tag element to specify how long to delay the next rollback, or omit that tag element to use the default of 10 minutes. The rollback can be delayed repeatedly in this way.

To commit the configuration immediately and permanently after emitting the `<confirmed/>` tag, emit the empty `<commit-configuration/>` tag before the rollback deadline passes. The device commits the candidate configuration and cancels the rollback. If the candidate configuration is still the same as the current committed configuration, the effect is the same as recommitting the current committed configuration.



NOTE: The confirmed commit operation is not available for a private copy of the configuration.

- On a device with two Routing Engines, commit the candidate configuration stored on the local Routing Engine on both Routing Engines. Combine tag elements as indicated in the following:
 - To copy the candidate configuration stored on the local Routing Engine to the other Routing Engine, verify the candidate's syntactic correctness, and commit it immediately on both Routing Engines, enclose the `<synchronize/>` tag in the `<commit-configuration>` tag element.
 - To copy the candidate configuration stored on the local Routing Engine to the other Routing Engine, verify the candidate's syntactic correctness, and commit it on both Routing Engines at a defined future time, enclose the `<synchronize/>` or `<force-synchronize/>` tag and `<at-time>` tag element in the `<commit-configuration>` tag element. Set the value in the `<at-time>` tag element as previously described for use of the `<at-time>` tag element alone.
 - To copy the candidate configuration stored on the local Routing Engine to the other Routing Engine and verify the candidate's syntactic correctness on each Routing Engine, enclose the `<synchronize/>` or `<force-synchronize/>` and `<check/>` tag elements in the `<commit-configuration>` tag element.
 - To copy the candidate configuration stored on the local Routing Engine to the other Routing Engine, verify the candidate's syntactic correctness, and commit it on both Routing Engines but require confirmation, enclose the `<synchronize/>` tag and `<confirmed/>` tag elements, and optionally the `<confirm-timeout>` tag element, in the `<commit-configuration>` tag element. Set the value in the `<confirm-timeout>`

tag element as previously described for use of the `<confirmed/>` tag and `<confirm-timeout>` tag element alone.

- To force the same synchronized commit operation as invoked by the `<synchronize/>` tag to succeed, even if there are open configuration sessions or uncommitted configuration changes on the remote machine, enclose the `<force-synchronize/>` tag in the `<commit-configuration>` tag element.
- To schedule the configuration for commit at a future time, enclose the `<at-time>` tag element in the `<commit-configuration>` tag element. There are three valid types of time specifiers:
 - The string **reboot**, to commit the configuration the next time the device reboots.
 - A time value of the form *hh:mm[:ss]* (hours, minutes, and, optionally, seconds), to commit the configuration at the specified time, which must be in the future but before 11:59:59 PM on the day the `<commit-configuration>` tag element is emitted. Use 24-hour time for the *hh* value; for example, 04:30:00 means 4:30:00 AM and 20:00 means 8:00 PM. The time is interpreted with respect to the clock and time zone settings on the device.
 - A date and time value of the form *yyyy-mm-dd hh:mm[:ss]* (year, month, date, hours, minutes, and, optionally, seconds), to commit the configuration at the specified date and time, which must be after the `<commit-configuration>` tag element is emitted. Use 24-hour time for the *hh* value. For example, 2005-08-21 15:30:00 means 3:30 PM on August 21, 2005. The time is interpreted with respect to the clock and time zone settings on the device.



NOTE: The time you specify must be more than 1 minute later than the current time on the device.

The configuration is checked immediately for syntactic correctness. If the check succeeds, the configuration is scheduled for commit at the specified time. If the check fails, the commit operation is not scheduled.

- Contents**
- `<at-time>`—Schedules the commit operation for a specified future time.
 - `<check>`—Requests verification that the configuration is syntactically correct, but does not actually commit it.
 - `<confirmed>`—Requests a commit of the candidate configuration and a rollback to the previous configuration after a short time, 10 minutes by default. Use the `<confirm-timeout>` tag element to specify a different amount of time.
 - `<confirm-timeout>`—Specifies the number of minutes for which the configuration remains active when the `<confirmed/>` tag is enclosed in the `<commit-configuration>` tag element.

`<log>`—Records a message in the file `/var/log/commits` when the commit operation succeeds.

`<synchronize>`—On dual control plane systems, requests that the candidate configuration on one control plane be copied to the other control plane, checked for correct syntax, and committed on both Routing Engines.

`<force-synchronize>`—On dual control plane systems, forces the candidate configuration on one control plane to be copied to the other control plane.

Related Documentation

- *Committing the Candidate Configuration the Using Junos XML Protocol*
- *Committing a Private Copy of the Configuration Using the Junos XML Protocol*
- *Committing a Configuration at a Specified Time Using the Junos XML Protocol*
- *Committing the Candidate Configuration Only After Confirmation Using the Junos XML Protocol*
- *Committing and Synchronizing a Configuration on Redundant Control Planes Using the Junos XML Protocol*
- *Logging a Message About a Commit Operation Using the Junos XML Protocol*
- [<commit-results> on page 149](#)
- [<open-configuration> on page 161](#)

<commit-results>

Usage

```
<rpc-reply xmlns:junos="URL">
  <!-- for the candidate configuration -->
  <commit-results>
    <routing-engine>...</routing-engine>
  </commit-results>

  <!-- for a private copy -->
  <commit-results>
    <load-success/>
    <routing-engine>...</routing-engine>
  </commit-results>

  <!-- for a private copy that does not include changes -->
  <commit-results>
  </commit-results>
</rpc-reply>
```

Release Information

This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description	NETCONF or Junos XML protocol server response containing information about a requested commit operation performed by the server on a particular Routing Engine.
Contents	<p><code><load-success/></code>—Indicates that the Junos XML protocol server successfully merged changes from the private copy into a copy of the candidate configuration, before committing the combined candidate on the specified Routing Engine.</p> <p>The <code><routing-engine></code> tag element is described separately.</p>
Related Documentation	<ul style="list-style-type: none">• <i>Committing the Candidate Configuration the Using Junos XML Protocol</i>• <commit-configuration> on page 145• <routing-engine> on page 162

`<database-status>`

Usage	<pre><xnm:error> <database-status-information> <database-status> <user>username</user> <terminal>terminal</terminal> <pid>pid</pid> <start-time>start-time</start-time> <idle-time>idle-time</idle-time> <commit-at>time</commit-at> <exclusive/> <edit-path>edit-path</edit-path> </database-status> </database-status-information> </xnm:error></pre>
Release Information	This is a Junos XML management protocol operation. It is a Juniper Networks proprietary extension to NETCONF and is identified in the capabilities exchange by the URI <code>http://xml.juniper.net/netconf/junos/1.0</code> . This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	Describe a user or NETCONF client application that is logged in to the configuration database. For simplicity, the Contents section uses the term user to refer to both human users and client applications, except where the information differs for the two.
Contents	<p><code><commit-at/></code>—Indicates that the user has scheduled a commit operation for a later time.</p> <p><code><edit-path></code>—Specifies the user's current location in the configuration hierarchy, in the form of the CLI configuration mode banner.</p> <p><code><exclusive/></code>—Indicates that the user or application has an exclusive lock on the configuration database. A user enters exclusive configuration mode by issuing the configure exclusive command in CLI operational mode. A client application obtains the lock by emitting the <code><lock-configuration/></code> tag element.</p>

`<idle-time>`—Specifies how much time has passed since the user last performed an operation in the database.

`<pid>`—Specifies the process ID of the Junos management process (mgd) that is handling the user's login session.

`<start-time>`—Specifies the time when the user logged in to the configuration database, in the format `YYYY-MM-DD hh:mm:ss TZ` (year, month, date, hour in 24-hour format, minute, second, time zone).

`<terminal>`—Identifies the UNIX terminal assigned to the user's connection.

`<user>`—Specifies the Junos OS login ID of the user whose login to the configuration database caused the error.

- Related Documentation**
- [<database-status-information> on page 151](#)
 - [<xnm:error> on page 164](#)

`<database-status-information>`

Usage

```
<xnm:error>
  <database-status-information>
    <database-status>...</database-status>
  </database-status-information>
</xnm:error>
]]>]]>
```

Release Information This is a Junos XML management protocol operation. It is a Juniper Networks proprietary extension to NETCONF and is identified in the capabilities exchange by the URI `http://xml.juniper.net/netconf/junos/1.0`. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Describe one or more users who have an open editing session in the configuration database.

The `<database-status>` tag element is explained separately.

- Related Documentation**
- [<database-status> on page 150](#)
 - [<xnm:error> on page 164](#)

`<end-session/>`

Usage

```
<rpc-reply xmlns:junos="URL">
  <end-session/>
</rpc-reply>
```

Release Information This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in

NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Indicate that the NETCONF or Junos XML protocol server is about to end the current session for a reason other than an error. Most often, the reason is that the client application has sent the `<request-end-session/>` tag.

Related Documentation

- *Ending a Junos XML Protocol Session and Closing the Connection*
- [<request-end-session/> on page 162](#)

<get-checksum-information>

Usage

```
<rpc>
  <get-checksum-information>
    <path>
      <!-- name and path of file -->
    </path>
  </get-checksum-information>
</rpc>
```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Request checksum information for the specified file.

Contents <path>—The name and path of the file to check.

Usage Guidelines See the *Junos XML API Operational Developer Reference*.

Related Documentation

- [<checksum-information> on page 144](#)

<get-configuration>

Usage

```
<rpc>
  <get-configuration
    [changed="changed"]
    [commit-scripts="( apply | apply-no-transients | view )"]
    [compare="rollback" [rollback="[0-49]"]]
    [database="(candidate | committed)"]
    [database-path=$junos-context/commit-context/database-path]
    [format="( json | text | xml )"]
    [inherit="( defaults | inherit )"
      [groups="groups" [interface-ranges="interface-ranges"]]
    [(junos:key | key)="key"] >
```

```

    <!-- tag elements for the configuration element to display -->
  </get-configuration>
</rpc>

```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Request configuration data from the NETCONF or Junos XML protocol server. The attributes specify the source and formatting of the data to display. Either the entire configuration hierarchy or a section can be displayed:

- To display the entire configuration hierarchy, emit the empty `<get-configuration/>` tag.
- To display a configuration element (hierarchy level or configuration object), emit tag elements within the `<get-configuration>` tag element to represent all levels of the configuration hierarchy from the root (represented by the `<configuration>` tag element) down to the level or object to display. To represent a hierarchy level or a configuration object that does not have an identifier, emit it as an empty tag. To represent an object that has one or more identifiers, emit its container tag element and identifier tag elements only, not any tag elements that represent other characteristics.



NOTE: Starting with Junos OS Release 13.1, within a NETCONF or Junos XML protocol session, a logical system user can use the Junos XML `<get-configuration>` operation to request specific logical system configuration hierarchies using child configuration tags as well as request the entire logical system configuration. When requesting the entire logical system configuration, the RPC reply includes the `<configuration>` root tag. Prior to Junos OS Release 13.1, the `<configuration>` root tag is omitted.

Attributes `changed`—Specifies that the `junos:changed="changed"` attribute should appear in the opening tag of each changed configuration element.

The attribute appears in the opening tag of every parent tag element in the path to the changed configuration element, including the top-level opening `<configuration>` tag. If the changed configuration element is represented by a single (empty) tag, the `junos:changed="changed"` attribute appears in the tag. If the changed element is represented by a container tag element, the `junos:changed="changed"` attribute appears in the opening container tag and also in each child tag element enclosed in the container tag element.

The `database` attribute can be combined with the `changed="changed"` attribute to request either the candidate or active configuration:

- When the candidate configuration is requested (the `database="candidate"` attribute is included or the `database` attribute is omitted completely), elements added to

the candidate configuration after the last commit operation are marked with the `junos:changed="changed"` attribute.

- When the active configuration is requested (the `database="committed"` attribute is included), elements added to the active configuration by the most recent commit are marked with the `junos:changed="changed"` attribute.



NOTE: When a commit operation succeeds, the server removes the `junos:changed="changed"` attribute from all tag elements. However, if warnings are generated during the commit, the attribute is not removed. In this case, the `junos:changed="changed"` attribute appears in tag elements that changed before the commit operation as well as on those that changed after it.

An example of a commit-time warning is the message explaining that a configuration element will not actually apply until the device is rebooted. The warning appears in the tag string that the server returns to confirm the success of the commit, enclosed in an `<xnm:warning>` tag element.

To remove the `junos:changed="changed"` attribute from elements that changed before the commit, take the action necessary to eliminate the cause of the warning, and commit the configuration again.

commit-scripts—Requests that the NETCONF or Junos XML protocol server display commit-script-style XML data. The value of the attribute determines the output. Acceptable values are:

- **apply**—Display the configuration with commit script changes applied, including both transient and non-transient changes. The output is equivalent to the CLI output when using the `| display commit-scripts` option. This attribute value is available starting with Junos OS Release 12.1.
- **apply-no-transients**—Display the configuration with commit script changes applied, but exclude transient changes. The output is equivalent to the CLI output when using the `| display commit-scripts no-transients` option. This attribute value is available starting with Junos OS Release 12.1.
- **view**—Display the configuration in the XML format that is input to a commit script. This is equivalent to viewing the configuration with the attributes `inherit="inherit"`, `groups="groups"`, and `changed="changed"`. The output is equivalent to the CLI output when using the `| display commit-scripts view` option.

compare—Requests that the NETCONF or Junos XML protocol server display the differences between the active or candidate configuration and a previously committed configuration. The only acceptable value for the **compare** attribute is **rollback**. The compare attribute is combined with the `rollback="rollback-number"` to specify which previously committed configuration should be used in the comparison. If the **rollback** attribute is omitted, the comparison uses rollback number 0, which is the active configuration.

The **database** attribute can be combined with the **compare="rollback"** attribute to request either the candidate or active configuration. If the **database** attribute is omitted, the candidate configuration is used. When the **compare** attribute is used, the default format for the output is text. If the client application attempts to include the **format="xml"** attribute when the **compare="rollback"** attribute is present, the server will return an `<xnm:error>` element indicating an error.

database—Specifies the version of the configuration from which to display data. There are two acceptable values:

- **candidate**—The candidate configuration.
- **committed**—The active configuration (the one most recently committed).

The **database** attribute takes precedence over the **database-path** attribute, if both are included.

database-path—Within a commit script, this attribute specifies the path to the session's pre-inheritance candidate configuration. The only acceptable value is **\$junos-context/commit-context/database-path**.

For normal configuration sessions, the commit script retrieves the normal, pre-inheritance candidate configuration. For private configuration sessions, the commit script retrieves the private, pre-inheritance candidate configuration. This attribute is available starting with Junos OS Release 12.2.

If you include both the **database** and the **database-path** attributes, the **database** attribute takes precedence.

format—Specifies the format in which the NETCONF or Junos XML protocol server returns the configuration data. Acceptable values are:

- **json**—Configuration statements are formatted in JavaScript Object Notation (JSON).
- **text**—Configuration statements are formatted as ASCII text, using the newline character, tabs and other white space, braces, and square brackets to indicate the hierarchical relationships between the statements. This is the format used in configuration files stored on a device running Junos OS and displayed by the CLI **show configuration** command.
- **xml**—Configuration statements are represented by the corresponding Junos XML tag elements. This is the default value if the **format** attribute is omitted.

groups—Specifies that the **junos:group="group-name"** attribute appears in the opening tag for each configuration element that is inherited from a configuration group. The *group-name* variable specifies the name of the configuration group.

The **groups** attribute must be combined with the **inherit** attribute, and the one acceptable value for it is **groups**.

inherit—Specifies how the NETCONF or Junos XML protocol server displays statements that are defined in configuration groups and interface ranges. If the **inherit** attribute is omitted, the output uses the `<groups>`, `<apply-groups>`, and `<apply-groups-except>` tag elements to represent user-defined configuration groups and uses the

<interface-range> tag element to represent user-defined interface ranges; it does not include tag elements for statements defined in the junos-defaults group.

There are two acceptable values:

- **defaults**—The output does not include the **<groups>**, **<apply-groups>**, and **<apply-groups-except>** tag elements, but instead displays tag elements that are inherited from user-defined groups and from the junos-defaults group as children of the inheriting tag elements.
- **inherit**—The output does not include the **<groups>**, **<apply-groups>**, **<apply-groups-except>**, and **<interface-range>** tag elements, but instead displays tag elements that are inherited from user-defined groups and ranges as children of the inheriting tag elements. The output does not include tag elements for statements defined in the junos-defaults group.

interface-ranges—Specifies that the **junos:interface-ranges="source-interface-range"** attribute appears in the opening tag for each configuration element that is inherited from an interface-range. The *source-interface-range* variable specifies the name of the interface-range.

The **interface-ranges** attribute must be combined with the **inherit** attribute, and the one acceptable value for it is **interface-ranges**.

junos:key | key—Specifies that the **junos:key="key"** attribute appears in the opening tag of each element that serves as an identifier for a configuration object. The only acceptable value is **key**.

Related Documentation

- *Requesting Configuration Data Using the Junos XML Protocol*
- *junos:changed*
- *junos:group*
- *junos:interface-range*
- *junos:key*
- *Junos XML API Configuration Developer Reference*

<load-configuration>

Usage	<pre><rpc> <load-configuration rescue="rescue"/> <load-configuration rollback="index"/> <load-configuration url="url" [action="(merge override replace update)] [format="(text xml)"] /> <load-configuration url="url" action="set" format="text"/> <load-configuration [action="(merge override replace update)] [format="xml"]></pre>
-------	--


```

<configuration>
  <!-- tag elements for configuration elements to load -->
</configuration>
</load-configuration>

<load-configuration [action="(merge | override | replace | update)"]
  format="text">
  <configuration-text>
    <!-- formatted ASCII configuration statements to load -->
  </configuration-text>
</load-configuration>

<load-configuration action="set" format="text">
  <configuration-set>
    <!-- configuration mode commands to load -->
  </configuration-set>
</load-configuration>
</rpc>

```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI `http://xml.juniper.net/netconf/junos/1.0` in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Request that the NETCONF or Junos XML protocol server load configuration data into the candidate configuration. Provide the data to load in one of the following ways :

- Set the empty `<load-configuration/>` tag's **rescue** attribute to the value **rescue**. The rescue configuration completely replaces the candidate configuration.
- Set the empty `<load-configuration/>` tag's **rollback** attribute to the numerical index of a previous configuration. The device stores a copy of the most recently committed configuration and up to 49 previous configurations. The specified previous configuration completely replaces the candidate configuration.
- Set the empty `<load-configuration/>` tag's **url** attribute to the pathname of a file that contains the configuration data to load. If providing the configuration data as formatted ASCII text, set the **format** attribute to **text**. If providing the configuration data as Junos XML tag elements, either omit the **format** attribute or set the value to **xml**. If providing the configuration data as a set of configuration mode commands, set the **action** attribute to **set**, and either omit the **format** attribute or set the value to **text**.

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

```
<load-configuration url="/tmp/add.conf"/>
```

- Enclose the configuration data within an opening `<load-configuration>` and closing `</load-configuration>` tag. If providing the configuration data as formatted ASCII text, enclose it in a `<configuration-text>` tag element, and set the **format** attribute to **text**. If providing the configuration data as Junos XML tag elements, enclose it in a `<configuration>` tag element, and either omit the **format** attribute or set the value to **xml**. If providing the configuration data as a set of configuration mode commands,

enclose it in a **<configuration-set>** tag element, set the **action** attribute to **set**, and either omit the **format** attribute or set the value to **text**.

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 for the corresponding level or object in the candidate configuration.

If providing the configuration data as formatted ASCII text (either in the file named by the **url** attribute or enclosed in a **<configuration-text>** tag element), also place the **replace:** statement on the line directly preceding the statements that represent the hierarchy level or object to replace. For more information, see the discussion of loading a file of configuration data in the *CLI User Guide*.

If providing the configuration data as Junos XML tag elements, 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.

- **set**—Loads a set of Junos OS configuration mode commands. This option executes the configuration instructions line by line as they are stored in a file named by the **url** attribute or enclosed in a **<configuration-set>** tag element. The instructions can contain any configuration mode command, such as **set**, **delete**, **edit**, or **deactivate**. When providing the configuration data as a set of commands, the only acceptable value for the **format** attribute is **text**. If the **action** attribute value is **set**, and the **format** attribute is omitted, the format automatically defaults to **text** rather than **xml**. This option was added in Junos OS Release 11.4.
- **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.

format—Specifies the format used for the configuration data. Acceptable values are:

- **text**—Indicates that configuration data is formatted as ASCII text or as a set of configuration mode commands.

ASCII text format uses the newline character, tabs and other white space, braces, and square brackets to indicate the hierarchical relationships between the

statements. This is the format used in configuration files stored on the routing platform and is displayed by the CLI **show configuration** command. Set command format consists of a series of Junos OS configuration mode commands and is displayed by the CLI **show configuration | display set** command. To import a set of configuration mode commands, you must set the **action** attribute to **set**.

- **xml**—Indicates that configuration statements are represented by the corresponding Junos XML tag elements. If the **format** attribute is omitted, **xml** is the default format for all values of the **action** attribute except **set**, which defaults to format **text**.

rescue—Specifies that the rescue configuration replace the current candidate configuration. The only valid value is **rescue**.

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 value can be a local file path, an FTP location, or a Hypertext Transfer Protocol (HTTP) URL:

- A local filename can have one of the following forms:
 - **/path/filename**—File on a mounted file system, either on the local flash disk or on hard disk.
 - **a:filename** or **a:path/filename**—File on the local drive. The default path is **/** (the root-level directory). The removable media can be in MS-DOS or UNIX (UFS) format.
- A filename on an FTP server has the following form:

ftp://username:password@hostname/path/filename
- A filename on an HTTP server has the following form:

http://username:password@hostname/path/filename

In each case, the default value for the **path** variable is the home directory for the username. To specify an absolute path, the application starts the path with the characters **%2F**; for example, **ftp://username:password@hostname/%2Fpath/filename**.

Related Documentation

- *Requesting Configuration Changes Using the Junos XML Protocol*
- [<load-configuration-results> on page 159](#)
- *replace*
- entries for **<configuration>**, **<configuration-text>**, and **<configuration-set>** in the *Junos XML API Configuration Developer Reference*

<load-configuration-results>

Usage `<rpc-reply xmlns:junos="URL">`

```
<load-configuration-results>
  <load-success/>
  <load-error-count>errors</load-error-count>
</load-configuration-results>
</rpc-reply>
```

Release Information	This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	NETCONF or Junos XML protocol server response tag returned after a <load-configuration> request by a client application. The tag encloses either a <load-success/> tag or a <load-error-count> tag, which indicates the success or failure of the load configuration operation.
Contents	<p><load-error-count>—Specifies the number of errors that occurred when the server attempted to load new data into the candidate configuration. The candidate configuration must be restored to a valid state before it is committed.</p> <p><load-success/>—Indicates that the server successfully loaded new data into the candidate configuration.</p>
Related Documentation	<ul style="list-style-type: none">• <load-configuration> on page 156

<lock-configuration/>

Usage	<pre><rpc> <lock-configuration/> </rpc></pre>
Release Information	This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	<p>Request that the NETCONF or Junos XML protocol server open and lock the candidate configuration, enabling the client application both to read and change it, but preventing any other users or applications from changing it. The application must emit the <unlock-configuration/> tag to unlock the configuration.</p> <p>If the Junos XML protocol session ends or the application emits the <unlock-configuration/> tag before the candidate configuration is committed, all changes made to the candidate are discarded.</p>
Related Documentation	<ul style="list-style-type: none">• <i>Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol</i>

- [<unlock-configuration/> on page 163](#)

<open-configuration>

Usage	<pre><rpc> <open-configuration> <private/> </open-configuration> </rpc></pre>
Release Information	<p>This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.</p>
Description	<p>Create a private copy of the candidate configuration.</p> <p>The client application can perform the same operations on the private copy as on the regular candidate configuration, including the commit operation. There are, however, restrictions on the commit operation. For details, see "<commit-configuration> on page 145."</p> <p>To discard a private copy without committing it, emit the empty <close-configuration/> tag. Changes to the private copy are also lost if the NETCONF or Junos XML protocol session ends for any reason before the changes are committed. It is not possible to save changes to a private copy other than by emitting the <commit-configuration/> tag.</p>
Related Documentation	<ul style="list-style-type: none"> • <i>Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol.</i> • <close-configuration/> on page 144 • <commit-configuration> on page 145 • <lock-configuration/> on page 160

<reason>

Usage	<pre><xnm:error xnm:warning> <reason> <daemon>process</daemon> <process-not-configured/> <process-disabled/> <process-not-running/> </reason> </xnm:error xnm:warning></pre>
Release Information	<p>This is a Junos XML management protocol operation. It is a Juniper Networks proprietary extension to NETCONF and is identified in the capabilities exchange by the URI</p>

<http://xml.juniper.net/netconf/junos/1.0> . This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Child element included in an `<xnm:error>` or `<xnm:warning>` element in a NETCONF protocol server response to explain why a process could not service a request.

Contents `<daemon>`—Identifies the process.

`<process-disabled>`—Indicates that the process has been explicitly disabled by an administrator.

`<process-not-configured>`—Indicates that the process has been disabled because it is not configured.

`<process-not-running>`—Indicates that the process is not running.

Related Documentation

- [<xnm:error> on page 164](#)
- [<xnm:warning> on page 165](#)

<request-end-session/>

Usage

```
<rpc>
  <request-end-session/>
</rpc>
```

Release Information This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI <http://xml.juniper.net/netconf/junos/1.0> in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.

Description Request that the NETCONF or Junos XML protocol server end the current session.

Related Documentation

- [<end-session/> on page 151](#)

<routing-engine>

Usage

```
<rpc-reply xmlns:junos="URL">
  <commit-results>

    <!-- when the candidate configuration is committed -->
    <routing-engine>
      <name>reX</name>
      <commit-success/>
    </routing-engine>

    <!-- when the candidate configuration is syntactically valid -->
    <routing-engine>
      <name>reX</name>
```

```

        <commit-check-success/>
    </routing-engine>

</commit-results>
</rpc-reply>

```

Release Information	This is a Junos XML management protocol response tag. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	Child element included in a NETCONF or Junos XML protocol server <commit-results> response element to return information about a requested commit operation on a particular Routing Engine.
Contents	<p><commit-check-success>—Indicates that the candidate configuration is syntactically correct.</p> <p><commit-success>—Indicates that the Junos XML protocol server successfully committed the candidate configuration.</p> <p><name>—Name of the Routing Engine on which the commit operation was performed. Possible values are re0 and re1.</p>
Related Documentation	<ul style="list-style-type: none"> • <commit-results> on page 149

[<unlock-configuration/>](#)

Usage	<pre> <rpc> <unlock-configuration/> </rpc> </pre>
Release Information	This is a Junos XML management protocol operation. It is supported in Junos XML protocol sessions, and it is supported as a Juniper Networks proprietary extension in NETCONF sessions that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	Request that the NETCONF or Junos XML protocol server unlock and close the candidate configuration. Until the application emits this tag, other users or applications can read the configuration but cannot change it.
Related Documentation	<ul style="list-style-type: none"> • <i>Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol</i> • <lock-configuration/> on page 160

<xnm:error>

Usage	<pre><xnm:error xmlns="<i>namespace-URL</i>" xmlns:xnm="<i>namespace-URL</i>"> <parse/> <source-daemon><i>module-name</i> </source-daemon> <filename><i>filename</i></filename> <line-number><i>line-number</i> </line-number> <column><i>column-number</i></column> <token><i>input-token-id</i> </token> <edit-path><i>edit-path</i></edit-path> <statement><i>statement-name</i> </statement> <message><i>error-string</i></message> <re-name><i>re-name-string</i></re-name> <database-status-information>...</database-status-information> <reason>...</reason> </xnm:error></pre>
Release Information	This is a Junos XML management protocol operation. It is a Juniper Networks proprietary extension to NETCONF and is identified in the capabilities exchange by the URI http://xml.juniper.net/netconf/junos/1.0 . This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.
Description	Indicate that the NETCONF server has experienced an error while processing the client application's request. If the server has already emitted the response tag element for the current request, the information enclosed in the response tag element might be incomplete. The client application must include code that discards or retains the information, as appropriate. The child tag elements described in the Contents section detail the nature of the error. The NETCONF server does not necessarily emit all child tag elements; it omits tag elements that are not relevant to the current request.
Attributes	<p>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 <i>version</i> is a string such as 1.1.</p> <p>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 <i>version</i> is a string such as 1.1.</p>
Contents	<p><column>—(Occurs only during loading of a configuration file) Identifies the element that caused the error by specifying its position as the number of characters after the first character in the specified line in the configuration file that was being loaded. The line and file are specified by the accompanying <line-number> and <filename> tag elements.</p> <p><edit-path>—(Occurs only during loading of configuration data) Specifies the path to the configuration hierarchy level at which the error occurred, in the form of the CLI configuration mode banner.</p> <p><filename>—(Occurs only during loading of a configuration file) Names the configuration file that was being loaded.</p>

<line-number>—(Occurs only during loading of a configuration file) Specifies the line number where the error occurred in the configuration file that was being loaded, which is named by the accompanying **<filename>** tag element.

<message>—Describes the error in a natural-language text string.

<parse/>—Indicates that there was a syntactic error in the request submitted by the client application.

<re-name>—Names the Routing Engine on which the error occurred.

<source-daemon>—Names the Junos OS module that was processing the request in which the error occurred.

<statement>—(Occurs only during loading of configuration data) Identifies the configuration statement that was being processed when the error occurred. The accompanying **<edit-path>** tag element specifies the statement's parent hierarchy level.

<token>—Names which element in the request caused the error.

The other tag elements are explained separately.

- Related Documentation**
- [<database-status-information> on page 151](#)
 - [<reason> on page 161](#)
 - [<xnm:warning> on page 165](#)

<xnm:warning>

Usage	<pre> <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</edit-path> <statement>statement-name </statement> <message>error-string</message> <reason>...</reason> </xnm:warning> </pre>
Release Information	<p>This is a Junos XML management protocol operation. It is a Juniper Networks proprietary extension to NETCONF and is identified in the capabilities exchange by the URI http://xml.juniper.net/netconf/junos/1.0. This operation is only supported in NETCONF sessions on Juniper Networks devices running Junos OS.</p>
Description	<p>Indicate that the server has encountered a problem while processing the client application's request. The child tag elements described in the Contents section detail the nature of the warning.</p>

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 in 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>**—(Occurs only during loading of a configuration file) Identifies the element that caused the problem by specifying its position as the number of characters after the first character in the specified line in the configuration file that was being loaded. The line and file are specified by the accompanying **<line-number>** and **<filename>** tag elements.

<edit-path>—(Occurs only during loading of configuration data) Specifies the path to the configuration hierarchy level at which the problem occurred, in the form of the CLI configuration mode banner.

<filename>—(Occurs only during loading of a configuration file) Names the configuration file that was being loaded.

<line-number>—(Occurs only during loading of a configuration file) Specifies the line number where the problem occurred in the configuration file that was being loaded, which is named by the accompanying **<filename>** tag element.

<message>—Describes the warning in a natural-language text string.

<source-daemon>—Names the Junos OS module that was processing the request in which the warning occurred.

<statement>—(Occurs only during loading of configuration data) Identifies the configuration statement that was being processed when the error occurred. The accompanying **<edit-path>** tag element specifies the statement's parent hierarchy level.

<token>—Names which element in the request caused the warning.

The other tag element is explained separately.

- Related Documentation**
- [<reason> on page 161](#)
 - [<xnm:error> on page 164](#)

Junos XML Protocol Element Attributes Supported in NETCONF Sessions

junos:changed-localtime

Usage	<pre><rpc-reply xmlns:junos="URL"> <configuration xmlns="URL" junos:changed-seconds="seconds" \ junos:changed-localtime="YYYY-MM-DD hh:mm:ss TZ"> <!-- Junos XML tag elements for the requested configuration data --> </configuration> </rpc-reply></pre>
Description	(Displayed when the candidate configuration is requested) Specify the time when the configuration was last changed as the date and time in the device's local time zone.
Usage Guidelines	See "Specifying the Source for Configuration Information Requests Using NETCONF" on page 186.
Related Documentation	<ul style="list-style-type: none"> • <configuration> in the <i>Junos XML API Configuration Developer Reference</i> • <rpc-reply> on page 141 • junos:changed-seconds on page 167 • xmlns on page 171

junos:changed-seconds

Usage	<pre><rpc-reply xmlns:junos="URL"> <configuration xmlns="URL" junos:changed-seconds="seconds" \ junos:changed-localtime="YYY-MM-DD hh:mm:ss TZ"> <!-- Junos XML tag elements for the requested configuration data --> </configuration> </rpc-reply></pre>
Description	(Displayed when the candidate configuration is requested) Specify the time when the configuration was last changed as the number of seconds since midnight on 1 January 1970.

Usage Guidelines See [“Specifying the Source for Configuration Information Requests Using NETCONF”](#) on page 186.

Related Documentation

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- `<rpc-reply>` on page 141
- `junos:changed-localtime` on page 167
- `xmlns` on page 171

`junos:commit-localtime`

Usage

```
<rpc-reply xmlns:junos="URL">
  <configuration xmlns="URL" junos:commit-seconds="seconds" \
    junos:commit-localtime="YYYY-MM-DD hh:mm:ss TZ" \
    junos:commit-user="username">
    <!-- Junos XML tag elements for the requested configuration data -->
  </configuration>
</rpc-reply>
```

Description (Displayed when the active configuration is requested) Specify the time when the configuration was committed as the date and time in the device's local time zone.

Usage Guidelines See [“Specifying the Source for Configuration Information Requests Using NETCONF”](#) on page 186.

Related Documentation

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- `<rpc-reply>` on page 141
- `junos:commit-user` on page 169
- `junos:commit-seconds` on page 168
- `xmlns` on page 171

`junos:commit-seconds`

Usage

```
<rpc-reply xmlns:junos="URL">
  <configuration xmlns="URL" junos:commit-seconds="seconds" \
    junos:commit-localtime="YYY-MM-DD hh:mm:ss TZ" \
    junos:commit-user="username">
    <!--Junos XML tag elements for the requested configuration data -->
  </configuration>
</rpc-reply>
```

Description (Displayed when the active configuration is requested) Specify the time when the configuration was committed as the number of seconds since midnight on 1 January 1970.

Usage Guidelines See [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186.](#)

Related Documentation

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- `<rpc-reply>` on page 141
- `junos:commit-user` on page 169
- `junos:commit-localtime` on page 168
- `xmlns` on page 171

junos:commit-user

Usage

```
<rpc-reply xmlns:junos="URL">
  <configuration xmlns="URL" junos:commit-seconds="seconds" \
    junos:commit-localtime="YYY-MM-DD hh:mm:ss TZ" \
    junos:commit-user="username">
    <!-- Junos XML tag elements for the requested configuration data -->
  </configuration>
</rpc-reply>
```

Description (Displayed when the active configuration is requested) Specify the Junos username of the user who requested the commit operation.

Usage Guidelines See [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186.](#)

Related Documentation

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- `<rpc-reply>` on page 141
- `junos:commit-localtime` on page 168
- `junos:commit-seconds` on page 168
- `xmlns` on page 171

operation

Usage

```
<rpc>
  <edit-config>
    <config>
      <configuration>
        <!-- opening tags for each parent of the changing element -->
        <changing-element operation="(create | delete | replace)">
          <name>identifier</name>
          <!-- if changing element has an identifier - -->
          <!-- other child tag elements, if appropriate for the operation -->
        </changing-element>
        <!-- closing tags for each parent of the changing element -->
      </configuration>
```

```
</config>
<!-- other child tag elements of the <edit-config> tag element -->
<edit-config>
</rpc>
]]>]]>
```

Description Specify how the NETCONF server incorporates an individual configuration element into the candidate configuration. If the attribute is omitted, the element is merged into the configuration according to the rules defined in [“Setting the Edit Configuration Mode in a NETCONF Session” on page 90](#). The following are acceptable values:

create—Creates the specified element in the configuration only if the element does not already exist.

delete—Deletes the specified element from the candidate configuration. We recommend that the **<default-operation>** tag element with the value **none** also be included in the **<edit-config>** tag element.

replace—Replaces the specified element in the candidate configuration with the provided new configuration data.

Related Documentation

- [Changing Individual Configuration Elements Using NETCONF on page 96](#)
- [Creating Configuration Elements Using NETCONF on page 99](#)
- [Deleting Configuration Elements Using NETCONF on page 101](#)
- [Replacing Configuration Elements Using NETCONF on page 106](#)
- [Setting the Edit Configuration Mode in a NETCONF Session on page 90](#)
- **<configuration>** in the *Junos XML API Configuration Developer Reference*
- **<edit-config>** on page 129

replace-pattern

Usage

```
<rpc>
<load-configuration>

<!-- replace a pattern globally -->
<configuration replace-pattern="pattern1" with="pattern2" [upto="n"]>
</configuration>

<!-- replace a pattern at a specific hierarchy level -->
<configuration>
  <!-- opening tag for each parent element -->
  <level-or-object replace-pattern="pattern1" with="pattern2" [upto="n"]/>
  <!-- closing tag for each parent element -->
</configuration>

<!-- replace a pattern for an object that has an identifier -->
<configuration>
  <!-- opening tag for each parent element -->
```

```

        <container-tag replace-pattern="pattern1" with="pattern2" [upto="n"]>
            <name>identifier</name>
        </container-tag>
        <!-- closing tag for each parent element -->
    </configuration>

</load-configuration>
</rpc>

```

Description	Replace a variable or identifier in the candidate configuration or open configuration database. Junos OS replaces the pattern specified by the replace-pattern attribute with the replacement pattern defined by the with attribute. The optional upto attribute limits the number of objects replaced. The scope of the replacement is determined by the placement of the attributes in the configuration data.
Attributes	<p>replace-pattern="pattern1"—Text string or regular expression that defines the identifiers or values you want to match.</p> <p>with="pattern2"—Text string or regular expression that replaces the identifiers and values located with <i>pattern1</i>.</p> <p>upto="n"—Number of objects replaced. The value of <i>n</i> controls the total number of objects that are replaced in the configuration (not the total number of times the pattern occurs). Objects at the same hierarchy level (siblings) are replaced first. Multiple occurrences of a pattern within a given object are considered a single replacement. If you do not include the upto attribute or you set the attribute equal to zero, all identifiers and values in the configuration that match the pattern are replaced.</p> <p>Range: 1 through 4294967295</p> <p>Default: 0</p>
Related Documentation	<ul style="list-style-type: none"> • Replacing Patterns in Configuration Data Using the NETCONF or Junos XML Protocol on page 108 • Using Global Replace in a Junos OS Configuration • Common Regular Expressions to Use with the replace Command • replace

xmlns

Usage	<pre> <rpc-reply xmlns:junos="URL"> <operational-response xmlns="URL-for-DTD"> <!-- Junos XML tag elements for the requested operational data --> </operational-response> </rpc-reply> <rpc-reply xmlns:junos="URL"> <configuration xmlns="URL" junos:(changed commit)-seconds="seconds" \ junos:(changed commit)-localtime="YYY-MM-DD hh:mm:ss TZ" \ [junos:commit-user="username"]> <!-- Junos XML tag elements for the requested configuration data --> </configuration> </pre>
--------------	--

`</rpc-reply>`

Description For operational responses, define the XML namespace for the enclosed tag elements that do not have a prefix (such as **junos:**) in their names. The namespace indicates which Junos XML document type definition (DTD) defines the set of tag elements in the response.

For configuration data responses, define the XML namespace for the enclosed tag elements.

Usage Guidelines See [“Requesting Operational Information Using NETCONF” on page 175](#) and [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

**Related
Documentation**

- `<configuration>` in the *Junos XML API Configuration Developer Reference*
- [<rpc-reply> on page 141](#)
- [junos:changed-localtime on page 167](#)
- [junos:changed-seconds on page 167](#)
- [junos:commit-user on page 169](#)
- [junos:commit-localtime on page 168](#)
- [junos:commit-seconds on page 168](#)

PART 4

Administration

- [Requesting Operational Information on page 175](#)
- [Requesting Configuration Information on page 183](#)
- [NETCONF Perl Client Applications on page 215](#)
- [YANG on page 235](#)

Requesting Operational Information

- Requesting Operational Information Using NETCONF on page 175
- Specifying the Output Format for Operational Information Requests in a NETCONF Session on page 177

Requesting Operational Information Using NETCONF

Within a NETCONF session, to request information about the current status of a device running Junos OS, a client application emits the specific tag element from the Junos XML API that returns the desired information. For example, the `<get-interface-information>` tag element corresponds to the `show interfaces` command, the `<get-chassis-inventory>` tag element requests the same information as the `show chassis hardware` command, and the `<get-system-inventory>` tag element requests the same information as the `show software information` command.

For complete information about the operational request tag elements available in the current Junos OS release, see “Mapping Between Operational Tag Elements, Perl Methods, and CLI Commands” and “Summary of Operational Request Tag Elements” in the *Junos XML API Operational Developer Reference*.

The application encloses the request tag in an `<rpc>` element. The syntax depends on whether the corresponding CLI command has any options.

```
<rpc>
  <!-- If the command does not have options -->
  <operational-request/>

  <!-- If the command has options -->
  <operational-request>
    <!-- tag elements representing the options -->
  </operational-request>
</rpc>
]]>]]>
```

The client application can specify the formatting of the information returned by the NETCONF server. By setting an optional attribute in the opening operational request tag, a client application can specify the format of the response as either XML-tagged format, which is the default, JavaScript Object Notation (JSON), or formatted ASCII text.

If the client application requests the output in formatted ASCII text, the NETCONF server encloses its response in an `<output>` tag element, which is enclosed in an `<rpc-reply>` tag.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <output>
    operational-response
  </output>
</rpc-reply>
]]>]]>
```

If the client application requests the output in formatted JSON, the NETCONF server encloses its response in the `<rpc-reply>` tag element.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  operational-response
</rpc-reply>
]]>]]>
```

If the client application requests the output in XML-tagged format, the NETCONF server encloses its response in the specific response tag element that corresponds to the request tag element, which is then enclosed in an `<rpc-reply>` tag element.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <operational-response xmlns="URL-for-DTD">
    <!-- tag elements for the requested information -->
  </operational-response>
</rpc-reply>
]]>]]>
```

For XML-tagged format, the opening tag for each operational response includes the `xmlns` attribute to define the XML namespace for the enclosed tag elements that do not have a prefix (such as `junos:`) in their names. The namespace indicates which Junos XML document type definition (DTD) defines the set of tag elements in the response. The Junos XML API defines separate DTDs for operational responses from different software modules. For instance, the DTD for interface information is called `junos-interface.dtd` and the DTD for chassis information is called `junos-chassis.dtd`. The division into separate DTDs and XML namespaces means that a tag element with the same name can have distinct functions depending on which DTD it is defined in.

The namespace is a URL of the following form:

```
http://xml.juniper.net/junos/release-code/junos-category
```

`release-code` is the standard string that represents the Junos OS release that is running on the NETCONF server device.

`category` specifies the DTD.

The *Junos XML API Operational Developer Reference* includes the text of the Junos XML DTDs for operational responses.

**Related
Documentation**

-

Specifying the Output Format for Operational Information Requests in a NETCONF Session

In a NETCONF session, to request information about a routing, switching, or security platform running Junos OS, a client application encloses a Junos XML request tag element in an `<rpc>` tag element. By setting the optional **format** attribute in the opening operational request tag, the client application can specify the formatting of the output returned by the NETCONF server. Information can be returned as JavaScript Object Notation (JSON), XML-tagged format, or formatted ASCII text. The basic syntax is as follows:

```
<rpc>
  <operational-request format="(xml | text | ascii | json)">
    <!-- tag elements for options -->
  </operational-request>
</rpc>
```

XML Format By default, the NETCONF server returns operational information in XML-tagged format. If the value of the **format** attribute is set to **xml**, or if the **format** attribute is omitted, the server returns the response in XML. The following example requests information for the ge-0/3/0 interface. The **format** attribute is omitted.

```
<rpc>
  <get-interface-information>
    <brief/>
    <interface-name>ge-0/3/0</interface-name>
  </get-interface-information>
</rpc>
]]>]]>
```



NOTE: Output of the configuration comparison in XML format using the **compare** filter is only supported from the CLI.

The NETCONF server returns the information in XML-tagged format, which is identical to the output displayed in the CLI when you include the **| display xml** option after the operational mode command.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/11.4R1/junos">
  <interface-information
    xmlns="http://xml.juniper.net/junos/11.4R1/junos-interface" junos:style="brief">

    <physical-interface>
      <name>ge-0/3/0</name>
      <admin-status junos:format="Enabled">up</admin-status>
      <oper-status>down</oper-status>
      <link-level-type>Ethernet</link-level-type>
      <mtu>1514</mtu>
      <source-filtering>disabled</source-filtering>
      <speed>1000mbps</speed>
      <bpdu-error>none</bpdu-error>
      <l2pt-error>none</l2pt-error>
      <loopback>disabled</loopback>
      <if-flow-control>enabled</if-flow-control>
```

```
<if-auto-negotiation>enabled</if-auto-negotiation>
<if-remote-fault>online</if-remote-fault>
<if-device-flags>
  <ifdf-present/>
  <ifdf-running/>
  <ifdf-down/>
</if-device-flags>
<if-config-flags>
  <iff-hardware-down/>
  <iff-snmp-traps/>
  <internal-flags>0x4000</internal-flags>
</if-config-flags>
<if-media-flags>
  <ifmf-none/>
</if-media-flags>
</physical-interface>
</interface-information>
</rpc-reply>
]]>]]>
```

JSON Format To request that the NETCONF server return operational information as formatted JSON instead of tagging it with Junos XML tag elements, the client application includes the **format="json"** attribute in the opening request tag. The client application encloses the request in an **<rpc>** tag element.

```
<rpc>
  <get-interface-information format="json">
    <brief/>
    <interface-name>cbp0</interface-name>
  </get-interface-information>
</rpc>
]]>]]>
```

When the client application includes the **format="json"** attribute in the request tag, the NETCONF server formats the reply using JSON.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/14.1I0/junos">
{
  "interface-information" : [
    {
      "attributes" : {"xmlns" :
"http://xml.juniper.net/junos/14.1I0/junos-interface",
      "junos:style" : "brief"
      },
      "physical-interface" : [
        {
          "name" : [
            {
              "data" : "cbp0"
            }
          ],
          "admin-status" : [
            {
              "data" : "up",
              "attributes" : {"junos:format" : "Enabled"}
            }
          ],
          "oper-status" : [
```

```

    {
      "data" : "up"
    },
    "if-type" : [
    {
      "data" : "Ethernet"
    }
    ],
    "link-level-type" : [
    {
      "data" : "Ethernet"
    }
    ],
    "mtu" : [
    {
      "data" : "1514"
    }
    ],
    "speed" : [
    {
      "data" : "Unspecified"
    }
    ],
    "clocking" : [
    {
      "data" : "Unspecified"
    }
    ],
    "if-device-flags" : [
    {
      "ifdf-present" : [
        {
          "data" : null
        }
      ],
      "ifdf-running" : [
        {
          "data" : null
        }
      ]
    }
    ],
    "ifd-specific-config-flags" : [
    {
    }
    ],
    "if-config-flags" : [
    {
      "iff-snmp-traps" : [
        {
          "data" : null
        }
      ]
    }
    ]
  }
]

```

```
}
</rpc-reply>
```

ASCII Format To request that the NETCONF server return operational information as formatted ASCII text instead of tagging it with Junos XML tag elements, the client application includes the **format="text"** or **format="ascii"** attribute in the opening request tag. The client application encloses the request in an **<rpc>** tag element.

```
<rpc>
  <get-interface-information format="(text | ascii)">
    <brief/>
    <interface-name>ge-0/3/0</interface-name>
  </get-interface-information>
</rpc>
]]>]]>
```

When the client application includes the **format="text"** or **format="ascii"** attribute in the request tag, the NETCONF server formats the reply as ASCII text and encloses it in an **<output>** tag element. The **format="text"** and **format="ascii"** attributes produce identical output.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:junos="http://xml.juniper.net/junos/11.4R1/junos">
  <output>
Physical interface: ge-0/3/0, Enabled, Physical link is Down
  Link-level type: Ethernet, MTU: 1514, Speed: 1000mbps, Loopback: Disabled,
  Source filtering: Disabled, Flow control: Enabled, Auto-negotiation: Enabled,
  Remote fault: Online
  Device flags   : Present Running Down
  Interface flags: Hardware-Down SNMP-Traps Internal: 0x4000
  Link flags     : None
  </output>
</rpc-reply>
]]>]]>
```

The following example shows the equivalent operational mode command executed in the CLI:

```
user@host> show interfaces ge-0/3/0 brief
Physical interface: ge-0/3/0, Enabled, Physical link is Down
  Link-level type: Ethernet, MTU: 1514, Speed: 1000mbps, Loopback: Disabled,
Source filtering: Disabled,
  Flow control: Enabled, Auto-negotiation: Enabled, Remote fault: Online
  Device flags   : Present Running Down
  Interface flags: Hardware-Down SNMP-Traps Internal: 0x4000
  Link flags     : None
```

The formatted ASCII text returned by the NETCONF server is identical to the CLI output except in cases where the output includes disallowed characters such as '<' (less-than sign), '>' (greater-than sign), and '&' (ampersand). The NETCONF server substitutes these characters with the equivalent predefined entity reference of '<', '>', and '&' respectively.

If the Junos XML API does not define a response tag element for the type of output requested by a client application, the NETCONF server returns the reply as formatted ASCII text enclosed in an **<output>** tag element, even if XML-tagged output is requested.

For information about the `<output>` tag element, see the *Junos XML API Operational Developer Reference*.



NOTE: The content and formatting of data within an `<output>` tag element are subject to change, so client applications must not depend on them.

CHAPTER 18

Requesting Configuration Information

- Requesting the Committed Configuration and Device State Using NETCONF on page 183
- Requesting Configuration Data Using NETCONF on page 184
- Specifying the Source for Configuration Information Requests Using NETCONF on page 186
- Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188
- Requesting the Complete Configuration Using NETCONF on page 188
- Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF on page 189
- Requesting All Configuration Objects of a Specified Type Using NETCONF on page 191
- Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF on page 194
- Requesting A Specific Configuration Object Using NETCONF on page 197
- Requesting Specific Child Tags for a Configuration Object Using NETCONF on page 199
- Requesting Multiple Configuration Elements Simultaneously Using NETCONF on page 203
- Requesting a Previous (Rollback) Configuration Using NETCONF on page 204
- Comparing Two Previous (Rollback) Configurations Using NETCONF on page 206
- Requesting the Rescue Configuration Using NETCONF on page 208
- Requesting an XML Schema for the Configuration Hierarchy Using NETCONF on page 210

Requesting the Committed Configuration and Device State Using NETCONF

In a NETCONF session with a device running Junos OS, to request the most recently committed configuration and the device state information for a routing, switching, or security platform, a client application encloses the **<get>** tag in an **<rpc>** tag element. By including the **<filter>** tag element and appropriate child tag elements, the application can request specific portions of the configuration. If the **<filter>** element is omitted, the server returns the entire configuration. The optional **format** attribute specifies the return format for the configuration data.

```
<rpc>  
  <get [format="(set | text | xml)"]>
```

```
<filter type="subtree">
  <!-- tag elements representing the configuration elements to return -->
</filter>
</get>
</rpc>
]]>]]>
```

The **type="subtree"** attribute in the opening **<filter>** tag indicates that the client application is using Junos XML tag elements to represent the configuration elements about which it is requesting information.

The NETCONF server encloses its reply in the **<rpc-reply>** and **<data>** tag elements. Within the **<data>** element, the configuration data is enclosed in the **<configuration>**, **<configuration-text>**, or **<configuration-set>** element depending on the requested format, and the device information is enclosed in the **<database-status-information>** element. The server includes attributes in the opening **<configuration>** tag that indicate the XML namespace for the enclosed tag elements and when the configuration was last changed or committed. For example:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" junos:changed-seconds="seconds"
      junos:changed-localtime="time">
      <!-- configuration data -->
    </configuration>
    <database-status-information>
      <database-status>
        <user>user</user>
        <terminal></terminal>
        <pid>pid</pid>
        <start-time junos:seconds="1416956595">2014-11-25 15:03:15 PST</start-time>
        <edit-path></edit-path>
      </database-status>
    </database-status-information>
  </data>
</rpc-reply>
]]>]]>
```

If there is no configuration data in the requested hierarchy, the RPC reply contains an empty **<configuration>** tag inside the **<data>** element.

- Related Documentation**
- [<get> on page 132](#)
 - [Requesting Configuration Data Using NETCONF on page 184](#)

Requesting Configuration Data Using NETCONF

In a NETCONF session with a device running Junos OS, to request configuration data for a routing, switching, or security platform, a client application encloses the **<get-config>**, **<source>**, and **<filter>** tag elements in an **<rpc>** tag element. By including the appropriate child tag element in the **<source>** tag element, the client application requests information from either the candidate or active configuration. By including the appropriate child tag

elements in the `<filter>` tag element, the application can request the entire configuration or specific portions of the configuration:

```
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
      <( candidate | running )/>
    </source>
    <filter type="subtree">
      <!-- tag elements representing the configuration elements to return -->
    </filter>
  </get-config>
</rpc>
]]>]]>
```

The `type="subtree"` attribute in the opening `<filter>` tag indicates that the client application is using Junos XML tag elements to represent the configuration elements about which it is requesting information.



NOTE: If the client application locks the candidate configuration before making requests, it needs to unlock it after making its read requests. Other users and applications cannot change the configuration while it remains locked. For more information, see [“Locking and Unlocking the Candidate Configuration Using NETCONF”](#) on page 72.

The NETCONF server encloses its reply in `<configuration>`, `<data>`, and `<rpc-reply>` tag elements. It includes attributes in the opening `<configuration>` tag that indicate the XML namespace for the enclosed tag elements and when the configuration was last changed or committed. For information about the attributes of the `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF”](#) on page 186.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- JUNOS XML tag elements representing configuration elements -->
    </configuration>
  </data>
</rpc-reply>
]]>]]>
```

If a Junos XML tag element is returned within an `<undocumented>` tag element, the corresponding configuration element is not documented in the Junos OS configuration guides or officially supported by Juniper Networks. Most often, the enclosed element is used for debugging only by support personnel. In a smaller number of cases, the element is no longer supported or has been moved to another area of the configuration hierarchy, but appears in the current location for backward compatibility.

Client applications can also request other configuration-related information, including an XML schema representation of the configuration hierarchy or information about previously committed configurations.

Related Documentation

- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Requesting a Previous \(Rollback\) Configuration Using NETCONF on page 204](#)
- [Comparing Two Previous \(Rollback\) Configurations Using NETCONF on page 206](#)
- [Requesting the Rescue Configuration Using NETCONF on page 208](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)
- [Requesting an XML Schema for the Configuration Hierarchy Using NETCONF on page 210](#)
- [Requesting Operational Information Using NETCONF on page 175](#)

Specifying the Source for Configuration Information Requests Using NETCONF

In a NETCONF session with a device running Junos OS, to request information from the candidate configuration, a client application includes the `<source>` element and `<candidate/>` tag within the `<rpc>` and `<get-config>` tag elements.

```
<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <!-- tag elements representing the configuration elements to return -->
    </filter>
  </get-config>
</rpc>
]]>]]>
```

To request information from the active configuration—the one most recently committed on the device—a client application includes the `<source>` tag element and `<running/>` tag enclosed within the `<rpc>` and `<get-config>` tag elements.

```
<rpc>
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <!-- tag elements representing the configuration elements to return -->
    </filter>
  </get-config>
</rpc>
]]>]]>
```



NOTE: If requesting the entire configuration, the application omits the `<filter>` tag element.

The NETCONF server encloses its reply in `<rpc-reply>`, `<data>`, and `<configuration>` tag elements. In the opening `<configuration>` tag, it includes the `xmlns` attribute to specify the namespace for the enclosed tag elements.

When returning information from the candidate configuration, the NETCONF server includes attributes that indicate when the configuration last changed (they appear on multiple lines here only for legibility).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" junos:changed-seconds="seconds" \
      junos:changed-localtime="YYYY-MM-DD hh:mm:ss TZ">
      <!-- Junos XML tag elements representing the configuration -->
    </configuration>
  </data>
</rpc-reply>
]>]]>
```

junos:changed-localtime represents the time of the last change as the date and time in the device's local time zone.

junos:changed-seconds represents the time of the last change as the number of seconds since midnight on 1 January 1970.

When returning information from the active configuration, the NETCONF server includes attributes that indicate when the configuration was committed (they appear on multiple lines here only for legibility).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" junos:commit-seconds="seconds" \
      junos:commit-localtime="YYYY-MM-DD hh:mm:ss TZ" \
      junos:commit-user="username">
      <!-- Junos XML tag elements representing the configuration -->
    </configuration>
  </data>
</rpc-reply>
]]>]]>
```

junos:commit-localtime represents the commit time as the date and time in the device's local time zone.

junos:commit-seconds represents the commit time as the number of seconds since midnight on 1 January 1970.

junos:commit-user specifies the Junos OS username of the user who requested the commit operation.

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [<get-config> on page 133](#)

Specifying the Scope of Configuration Information to Return in a NETCONF Response

In a NETCONF session with a device running Junos OS, a client application can request the entire configuration or specific portions of the configuration by including the appropriate child tag elements in the `<filter>` tag element within the `<rpc>` and `<get-config>` tag elements.

```
<rpc>
  <get-config>
    <source>
      ( <candidate/> | <running/> )
    </source>
    <filter>
      <!-- tag elements representing the configuration elements to return -->
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For information about requesting different amounts of configuration information, see the following topics:

- [Requesting the Complete Configuration Using NETCONF on page 188](#)
- [Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF on page 189](#)
- [Requesting All Configuration Objects of a Specified Type Using NETCONF on page 191](#)
- [Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF on page 194](#)
- [Requesting A Specific Configuration Object Using NETCONF on page 197](#)
- [Requesting Specific Child Tags for a Configuration Object Using NETCONF on page 199](#)
- [Requesting Multiple Configuration Elements Simultaneously Using NETCONF on page 203](#)

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)

Requesting the Complete Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, to request the entire candidate configuration, a client application encloses `<get-config>` and `<source>` tag elements and the `<candidate/>` tag in an `<rpc>` tag element:

```
<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
  </get-config>
</rpc>
```



```

    </source>
  </get-config>
</rpc>
]]>]]>

```

To request the entire active configuration, a client application encloses `<get-config>` and `<source>` tag elements and the `<running/>` tag in an `<rpc>` tag element:

```

<rpc>
  <get-config>
    <source>
      <running/>
    </source>
  </get-config>
</rpc>
]]>]]>

```

The NETCONF server encloses its reply in `<configuration>`, `<data>`, and `<rpc-reply>` tag elements. For information about the attributes in the opening `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF”](#) on page 186.

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- Junos XML tag elements representing the configuration -->
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

Related Documentation

- [Requesting Configuration Data Using NETCONF](#) on page 184
- [Specifying the Source for Configuration Information Requests Using NETCONF](#) on page 186
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response](#) on page 188
- [Requesting the Rescue Configuration Using NETCONF](#) on page 208
- [Requesting an XML Schema for the Configuration Hierarchy Using NETCONF](#) on page 210

Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF

In a NETCONF session with a device running Junos OS, to request complete information about all child configuration elements at a hierarchy level or in a container object that does not have an identifier, a client application emits a `<filter>` tag element that encloses the tag elements representing all levels in the configuration hierarchy from the root (represented by the `<configuration>` tag element) down to the immediate parent level of the level or container object, which is represented by an empty tag. The entire request is enclosed in an `<rpc>` tag element:

```

<rpc>

```

```
<get-config>
  <source>
    <!-- tag specifying the source configuration -->
  </source>
  <filter type="subtree">
    <configuration>
      <!-- opening tags for each parent of the requested level -->
      <level-or-container/>
      <!-- closing tags for each parent of the requested level -->
    </configuration>
  </filter>
</get-config>
</rpc>
]]>]]>
```

For information about the `<source>` tag element, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

The NETCONF server returns the requested section of the configuration in `<data>` and `<rpc-reply>` tag elements. For information about the attributes in the opening `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- opening tags for each parent of the level -->
      <level-or-container>
        <!-- child tag elements of the level or container -->
      </level-or-container>
      <!-- closing tags for each parent of the level -->
    </configuration>
  </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same `<get-config>` tag element. For more information, see [“Requesting Multiple Configuration Elements Simultaneously Using NETCONF” on page 203](#).

The following example shows how to request the contents of the `[edit system login]` hierarchy level in the candidate configuration.

Client Application

```

<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
  <filter>
    <configuration>
      <system>
        <login/>
      </system>
    </configuration>
  </filter>
</get-config>
</rpc>
]]>]]>

```

NETCONF Server

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <system>
        <login>
          <user>
            <name>barbara</name>
            <full-name>Barbara Anderson</full-name>
            <class>superuser</class>
            <uid>632</uid>
          </user>
          <!-- other child tag elements of <login> -->
        </login>
      </system>
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

T2128

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF on page 194](#)
- [Requesting Multiple Configuration Elements Simultaneously Using NETCONF on page 203](#)

Requesting All Configuration Objects of a Specified Type Using NETCONF

In a NETCONF session with a device running Junos OS, to request information about all configuration objects of a specified type in a hierarchy level, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels in the

configuration hierarchy from the root (represented by the `<configuration>` tag element) down to the immediate parent level for the object type. An empty tag returns all configuration objects of the requested object type and all child tags for each object. To return only specific child tags for the configuration objects, enclose the desired child tags in the opening and closing tags of the object. The entire request is enclosed in an `<rpc>` tag element:

```
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <!-- opening tags for each parent of the requested object type -->
        <object-type>
          <!-- optionally select specific child tags -->
          </object-type>
        <!-- closing tags for each parent of the requested object type -->
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For information about the `<source>` tag element, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

This type of request is useful when the object's parent hierarchy level has more than one type of child object. If the requested object is the only child type that can occur in its parent hierarchy level, then this type of request yields the same output as a request for the complete parent hierarchy, which is described in [“Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF” on page 189](#).

The NETCONF server returns the requested objects in `<data>` and `<rpc-reply>` tag elements. For information about the attributes in the opening `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- opening tags for each parent of the object type -->
      <first-object>
        <!-- child tag elements for the first object -->
      </first-object>
      <second-object>
        <!-- child tag elements for the second object -->
      </second-object>
      <!-- additional instances of the object -->
      <!-- closing tags for each parent of the object type -->
    </configuration>
  </data>
</rpc-reply>
```

```
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same `<get-config>` tag element. For more information, see [“Requesting Multiple Configuration Elements Simultaneously Using NETCONF” on page 203](#).

The following example shows how to request complete information about all **radius-server** objects at the **[edit system]** hierarchy level in the candidate configuration.

Client Application

NETCONF Server

```
<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <configuration>
        <system>
          <radius-server/>
        </system>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <system>
        <radius-server>
          <name>10.25.34.166</name>
          <secret>$9$Pf3900REcr/9t...</secret>
          <timeout>5</timeout>
          <retry>3</retry>
        </radius-server>
        <radius-server>
          <name>10.25.6.204</name>
          <secret>$9$K5Kvxd2gJZUi-d...</secret>
          <timeout>5</timeout>
          <retry>3</retry>
        </radius-server>
      </system>
    </configuration>
  </data>
</rpc-reply>
]]>]]>
```

T2129

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)

- [Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF on page 194](#)

Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF

In a NETCONF session with a device running Junos OS, to request output that shows only the identifier for each configuration object of a specific type in a hierarchy, a client application emits a `<filter>` tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the `<configuration>` tag element) down to the immediate parent level for the object type. The object type is represented by its container tag element enclosing an empty `<name/>` tag. (The `<name>` tag element can always be used, even if the actual identifier tag element has a different name. The actual name is also valid.) The entire request is enclosed in an `<rpc>` tag element:

```
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <!-- opening tags for each parent of the object type -->
        <object-type>
          <name/>
        </object-type>
        <!-- closing tags for each parent of the object type -->
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For information about the `<source>` tag element, see ["Specifying the Source for Configuration Information Requests Using NETCONF" on page 186](#).



NOTE: You cannot request only identifiers for object types that have multiple identifiers. However, for many such objects the identifiers are the only child tag elements, so requesting complete information yields the same output as requesting only identifiers. For instructions, see ["Requesting All Configuration Objects of a Specified Type Using NETCONF" on page 191](#).

The NETCONF server returns the requested objects in `<data>` and `<rpc-reply>` tag elements (here, objects for which the identifier tag element is called `<name>`). For information about the attributes in the opening `<configuration>` tag, see ["Specifying the Source for Configuration Information Requests Using NETCONF" on page 186](#).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
```

```

<!-- opening tags for each parent of the object type -->
<first-object>
  <name>identifier-for-first-object</name>
</first-object>
<second-object>
  <name>identifier-for-second-object</name>
</second-object>
<!-- additional objects -->
<!-- closing tags for each parent of the object type -->
</configuration>
</data>
</rpc-reply>
]]>]]>

```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see [“Requesting Multiple Configuration Elements Simultaneously Using NETCONF” on page 203](#).

The following example shows how to request the identifier for each BGP neighbor configured at the **[edit protocols bgp group next-door-neighbors]** hierarchy level in the candidate configuration.

Client Application

```

<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <configuration>
        <protocols>
          <bgp>
            <group>
              <name>next-door-neighbors</name>
              <neighbor>
                <name/>
              </neighbor>
            </group>
          </bgp>
        </protocols>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>

```

NETCONF Server

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <protocols>
        <bgp>
          <group>
            <name>next-door-neighbors</name>
            <neighbor>
              <name>10.2.35.188</name>
            </neighbor>
            <neighbor>
              <name>10.3.62.95</name>
            </neighbor>
            <neighbor>
              <name>10.4.122.9</name>
            </neighbor>
          </group>
        </bgp>
      </protocols>
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

T2130

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)
- [Requesting All Configuration Objects of a Specified Type Using NETCONF on page 191](#)

Requesting A Specific Configuration Object Using NETCONF

In a NETCONF session with a device running Junos OS, to request complete information about a specific configuration object, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object.

To represent the requested object, the application emits only the container tag element and each of its identifier tag elements, complete with identifier value, for the object. For objects with a single identifier, the **<name>** tag element can always be used, even if the actual identifier tag element has a different name. The actual name is also valid. For objects with multiple identifiers, the actual names of the identifier tag elements must be used. To verify the name of each of the identifiers for a configuration object, see the *Junos XML API Configuration Developer Reference*. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
  <get-config>
    <source>
      <!--tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <!-- opening tags for each parent of the object -->
        <object>
          <name>identifier</name>
        </object>
        <!-- closing tags for each parent of the object -->
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

The NETCONF server returns the requested object in **<data>** and **<rpc-reply>** tag elements (here, an object for which the identifier tag element is called **<name>**). For information about the attributes in the opening **<configuration>** tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- opening tags for each parent of the object -->
      <object>
        <name>identifier</name>
        <!-- other child tag elements of the object -->
      </object>
      <!-- closing tags for each parent of the object -->
    </configuration>
```

```

    </data>
  </rpc-reply>
]>]]>

```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same `<get-config>` tag element. For more information, see [“Requesting Multiple Configuration Elements Simultaneously Using NETCONF” on page 203](#).

The following example shows how to request the contents of one multicasting scope called `local`, which is at the `[edit routing-options multicast]` hierarchy level in the candidate configuration. To specify the desired object, the client application emits the `<name>local</name>` identifier tag element as the innermost tag element.

Client Application

```

<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <configuration>
        <routing-options>
          <multicast>
            <scope>
              <name>local</name>
            </scope>
          </multicast>
        </routing-options>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>

```

NETCONF Server

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <routing-options>
        <multicast>
          <scope>
            <name>local</name>
            <prefix>239.255.0.0/16</prefix>
            <interface>ip-f/p/0</interface>
          </scope>
        </multicast>
      </routing-options>
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

T2131

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)

- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)
- [Requesting Specific Child Tags for a Configuration Object Using NETCONF on page 199](#)

Requesting Specific Child Tags for a Configuration Object Using NETCONF

In a NETCONF session with a device running Junos OS, to request specific child tag elements and descendents for configuration objects, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object. To represent the requested object, the application emits its container tag element. To request a specific configuration object, include the identifier tag element. For objects with a single identifier, the **<name>** tag element can always be used, even if the actual identifier tag element has a different name. The actual name is also valid. For objects with multiple identifiers, the actual names of the identifier tag elements must be used. If you omit the identifier tag element, the server returns the child tags for all configuration objects of that type. To select specific child tags, the client application emits all desired child tag elements and descendents within the container tag element. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <!-- opening tags for each parent of the object -->
        <object>
          <name>identifier</name>
          <first-child/>
          <second-child/>
          <third-child>
            <!--tags for descendents-->
          </third-child>
          <!-- tag for each additional child to return -->
        </object>
        <!-- closing tags for each parent of the object -->
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

The NETCONF server returns the requested children of the object in **<data>** and **<rpc-reply>** tag elements (here, an object for which the identifier tag element is called **<name>**). For information about the attributes in the opening **<configuration>** tag, see

[“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186.](#)

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration attributes>
      <!-- opening tags for each parent of the object -->
      <object>
        <name>identifier</name>
      </object>
      <!-- closing tags for each parent of the object -->
    </configuration>
  </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see [“Requesting Multiple Configuration Elements Simultaneously Using NETCONF” on page 203.](#)

The following example shows how to request only the address of the next-hop device for the 192.168.5.0/24 route at the **[edit routing-options static]** hierarchy level in the candidate configuration.

Client Application

```

<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <configuration>
        <routing-options>
          <static>
            <route>
              <name>192.168.5.0/24</name>
              <next-hop/>
            </route>
          </static>
        </routing-options>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>

```

NETCONF Server

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <routing-options>
        <static>
          <route>
            <name>192.168.5.0/24</name>
            <next-hop>192.168.71.254</next-hop>
          </route>
        </static>
      </routing-options>
    </configuration>
  </data>
</rpc-reply>
]]>]]>

```

T2132

The following example shows how to request the addresses for all logical interfaces configured for each physical interface within the groups hierarchy level of the candidate configuration.

```

<rpc>
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter type="subtree">
      <configuration>
        <groups>
          <interfaces>
            <interface>
              <unit>
                <family>
                  <inet>
                    <address/>
                  </inet>
                </unit>
              </interface>
            </interfaces>
          </groups>
        </configuration>
      </filter>
    </get-config>
  </rpc>

```

```
        </family>
      </unit>
    </interface>
  </interfaces>
</groups>
</configuration>
</filter>
</get-config>
</rpc>

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <data>
    <configuration xmlns="URL" junos:commit-seconds=seconds
junos:commit-localtime="timestamp" junos:commit-user="user">
      <groups>
        <name>re0</name>
        <interfaces>
          <interface>
            <name>lo0</name>
            <unit>
              <name>0</name>
              <family>
                <inet>
                  <address>
                    <name>127.0.0.1/32</name>
                  </address>
                </inet>
              </family>
            </unit>
          </interface>
          <interface>
            <name>em0</name>
            <unit>
              <name>0</name>
              <family>
                <inet>
                  <address>
                    <name>198.51.100.1/24</name>
                  </address>
                  <address>
                    <name>198.51.100.11/24</name>
                  </address>
                </inet>
              </family>
            </unit>
          </interface>
        </interfaces>
      </groups>
    </configuration>
  </data>
</rpc-reply>
```

Related Documentation

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)

- [Requesting A Specific Configuration Object Using NETCONF on page 197](#)
- [Requesting Multiple Configuration Elements Simultaneously Using NETCONF on page 203](#)

Requesting Multiple Configuration Elements Simultaneously Using NETCONF

In a NETCONF session with a device running Junos OS, a client application can request multiple configuration elements of the same type or different types within a `<get-config>` tag element. The request includes only one `<filter>` and `<configuration>` tag element (the NETCONF server returns an error if there is more than one of each).

If two requested objects have the same parent hierarchy level, the client can either include both requests within one parent tag element, or repeat the parent tag element for each request. For example, at the `[edit system]` hierarchy level the client can request the list of configured services and the identifier tag element for RADIUS servers in either of the following two ways:

```
<!-- both requests in one <system> tag element -->
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <system>
          <services/>
          <radius-server>
            <name/>
          </radius-server>
        </system>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
<!-- separate <system> tag element for each element -->
<rpc>
  <get-config>
    <source>
      <!-- tag specifying the source configuration -->
    </source>
    <filter type="subtree">
      <configuration>
        <system>
          <services/>
        </system>
        <system>
          <radius-server>
            <name/>
          </radius-server>
        </system>
      </configuration>
    </filter>
  </get-config>
</rpc>
]]>]]>
```

```
        </configuration>
      </filter>
    </get-config>
  </rpc>
]>]]>
```

The client can combine requests for any of the following types of information:

- [Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using NETCONF on page 189](#)
- [Requesting All Configuration Objects of a Specified Type Using NETCONF on page 191](#)
- [Requesting Identifiers for Configuration Objects of a Specified Type Using NETCONF on page 194](#)
- [Requesting A Specific Configuration Object Using NETCONF on page 197](#)
- [Requesting Specific Child Tags for a Configuration Object Using NETCONF on page 199](#)

**Related
Documentation**

- [Requesting Configuration Data Using NETCONF on page 184](#)
- [Specifying the Source for Configuration Information Requests Using NETCONF on page 186](#)
- [Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188](#)

Requesting a Previous (Rollback) Configuration Using NETCONF

In a NETCONF session with a device running Junos OS, to request a previously committed (rollback) configuration, a client application emits the Junos XML

`<get-rollback-information>` tag element and its child `<rollback>` tag element in an `<rpc>` tag element. This operation is equivalent to the **show system rollback** operational mode command. The `<rollback>` tag element specifies the index number of the previous configuration to display; its value can be from 0 (zero, for the most recently committed configuration) through 49.

To request Junos XML-tagged output, the application either includes the `<format>` tag element with the value `xml` or omits the `<format>` tag element (Junos XML tag elements are the default):

```
<rpc>
  <get-rollback-information>
    <rollback> index-number </rollback>
  </get-rollback-information>
</rpc>
]>]]>
```

The NETCONF server encloses its response in `<rpc-reply>`, `<rollback-information>`, and `<configuration>` tag elements. The `<ok/>` tag is a side effect of the implementation and does not affect the results. For information about the attributes in the opening `<configuration>` tag, see ["Specifying the Source for Configuration Information Requests Using NETCONF" on page 186](#).


```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rollback-information>
    <ok/>
    <configuration attributes>
      <!-- tag elements representing the complete previous configuration -->
    </configuration>
  </rollback-information>
</rpc-reply>
]]>]]>

```

To request formatted ASCII output, the application includes the **<format>** tag element with the value **text**:

```

<rpc>
  <get-rollback-information>
    <rollback>index-number</rollback>
    <format>text</format>
  </get-rollback-information>
</rpc>
]]>]]>

```

The NETCONF server encloses its response in **<rpc-reply>**, **<rollback-information>**, **<configuration-information>**, and **<configuration-output>** tag elements. For more information about the formatted ASCII notation used in Junos OS configuration statements, see the *CLI User Guide*.

```

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rollback-information>
    <ok/>
    <configuration-information>
      <configuration-output>
        /* formatted ASCII representing the complete previous configuration */
      </configuration-output>
    </configuration-information>
  </rollback-information>
</rpc-reply>
]]>]]>

```

The following example shows how to request Junos XML-tagged output for the rollback configuration that has an index of 2. In actual output, the *Junos-version* variable is replaced by a value such as 15.1R1 for the initial version of Junos OS Release 15.1.

Client Application

```
<rpc>
  <get-rollback-information>
    <rollback>2</rollback>
  </get-rollback-information>
</rpc>
]]>]]>
```

NETCONF Server

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rollback-information>
    <ok/>
    <configuration xmlns="URL" \
      junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
      <version>JUNOS-version</version>
      <system>
        <host-name>big-router</host-name>
        <!-- other children of <system> -->
      </system>
      <!-- other children of <configuration> -->
    </configuration>
  </rollback-information>
</rpc-reply>
]]>]]>
```

T2133

Related Documentation

- [Comparing Two Previous \(Rollback\) Configurations Using NETCONF on page 206](#)
- [Requesting the Rescue Configuration Using NETCONF on page 208](#)

Comparing Two Previous (Rollback) Configurations Using NETCONF

In a NETCONF session with a device running Junos OS, to compare the contents of two previously committed (rollback) configurations, a client application emits the Junos XML **<get-rollback-information>** tag element and its child **<rollback>** and **<compare>** tag elements in an **<rpc>** tag element. This operation is equivalent to the **show system rollback** operational mode command with the **compare** option. The **<rollback>** tag element specifies the index number of the configuration that is the basis for comparison. The **<compare>** tag element specifies the index number of the configuration to compare with the base configuration. Valid values in both tag elements range from 0 (zero, for the most recently committed configuration) through 49:

```
<rpc>
  <get-rollback-information>
    <rollback>index-number</rollback>
    <compare>index-number</compare>
  </get-rollback-information>
</rpc>
]]>]]>
```



NOTE: The output corresponds more logically to the chronological order of changes if the older configuration (the one with the higher index number) is the base configuration. Its index number is enclosed in the `<rollback>` tag element and the index of the more recent configuration is enclosed in the `<compare>` tag element.

The NETCONF server encloses its response in `<rpc-reply>`, `<rollback-information>`, `<configuration-information>`, and `<configuration-output>` tag elements. The `<ok/>` tag is a side effect of the implementation and does not affect the results.

The information in the `<configuration-output>` tag element is formatted ASCII and includes a banner line (such as `[edit interfaces]`) for each hierarchy level at which the two configurations differ. Each line between banner lines begins with either a plus sign (+) or a minus sign (–). The plus sign indicates that adding the statement to the base configuration results in the second configuration, whereas a minus sign means that removing the statement from the base configuration results in the second configuration.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rollback-information>
    <ok/>
    <configuration-information>
      <configuration-output>
        /* formatted ASCII representing the changes */
      </configuration-output>
    </configuration-information>
  </rollback-information>
</rpc-reply>
]]>]]>
```

The following example shows how to request a comparison of the rollback configurations that have indexes of 20 and 4.

Client Application	NETCONF Server
<pre> <rpc> <get-rollback-information> <rollback>20</rollback> <compare>4</compare> </get-rollback-information> </rpc>]]>]]> </pre>	<pre> <rpc-reply xmlns="URN" xmlns:junos="URL"> <rollback-information> <ok/> <configuration-information> <configuration-output> [edit interfaces] - ge-0/2/0 { - stacked-vlan-tagging; - mac 00.01.02.03.04.05; - gigeather-options { - loopback; - } - } [edit] + services { + l2tp { + tunnel-group 12 { + local-gateway; + } + } + } </configuration-output> </configuration-information> </rollback-information> </rpc-reply>]]>]]> </pre>

T2117

- Related Documentation**
- [Requesting a Previous \(Rollback\) Configuration Using NETCONF on page 204](#)
 - [Requesting the Rescue Configuration Using NETCONF on page 208](#)

Requesting the Rescue Configuration Using NETCONF

The rescue configuration is a configuration saved in case it is necessary to restore a valid, nondefault configuration. (To create a rescue configuration in a NETCONF session, use the Junos XML **<request-save-rescue-configuration>** tag element or the **request system configuration rescue save** CLI operational mode command. For more information, see the *Junos XML API Operational Developer Reference* or the [CLI Explorer](#).)

In a NETCONF session with a device running Junos OS, a client application requests the rescue configuration by emitting the Junos XML **<get-rescue-information>** tag element in an **<rpc>** tag element. This operation is equivalent to the **show system configuration rescue** operational mode command.

To request Junos XML-tagged output, the application either includes the **<format>** tag element with the value **xml** or omits the **<format>** tag element (Junos XML tag elements are the default):

```
<rpc>
  <get-rescue-information/>
</rpc>
]]>]]>
```

The NETCONF server encloses its response in `<rpc-reply>`, `<rescue-information>`, and `<configuration>` tag elements. The `<ok/>` tag is a side effect of the implementation and does not affect the results. For information about the attributes in the opening `<configuration>` tag, see [“Specifying the Source for Configuration Information Requests Using NETCONF” on page 186](#).

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rescue-information>
    <ok/>
    <configuration attributes
      <!-- tag elements representing the rescue configuration -->
    </configuration>
  </rescue-information>
</rpc-reply>
]]>]]>
```

To request formatted ASCII output, the application includes the `<format>` tag element with the value `text`:

```
<rpc>
  <get-rescue-information>
    <format>text</format>
  </get-rescue-information>
</rpc>
]]>]]>
```

The NETCONF server encloses its response in `<rpc-reply>`, `<rescue-information>`, `<configuration-information>`, and `<configuration-output>` tag elements. For more information about the formatted ASCII notation used in Junos configuration statements, see the *CLI User Guide*.

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <rescue-information>
    <ok/>
    <configuration-information>
      <configuration-output>
        /* formatted ASCII for the rescue configuration*/
      </configuration-output>
    </configuration-information>
  </rescue-information>
</rpc-reply>
]]>]]>
```

Related Documentation

- [Requesting a Previous \(Rollback\) Configuration Using NETCONF on page 204](#)
- [Comparing Two Previous \(Rollback\) Configurations Using NETCONF on page 206](#)

Requesting an XML Schema for the Configuration Hierarchy Using NETCONF

The schema represents all configuration elements available in the version of the Junos OS that is running on a device. (To determine the Junos OS version, emit the **<get-software-information>** operational request tag element, which is documented in the *Junos XML API Operational Developer Reference*.)

Client applications can use the schema to validate the configuration on a device or simply to learn which configuration statements are available in the version of the Junos OS running on the device. The schema does not indicate which elements are actually configured or even that an element can be configured on that type of device (some configuration statements are available only on certain device types). To request the set of currently configured elements and their settings, emit the **<get-config>** tag element instead, as described in “Requesting Configuration Data Using NETCONF” on page 184.

Explaining the structure and notational conventions of the XML Schema language is beyond the scope of this document. For information, see *XML Schema Part 0: Primer*, available from the World Wide Web Consortium (W3C) at <http://www.w3.org/TR/xmlschema-0/>. The primer provides a basic introduction and lists the formal specifications where you can find detailed information.

For further information, see the following sections:

- [Requesting an XML Schema for the Configuration Hierarchy on page 210](#)
- [Creating the junos.xsd File on page 211](#)
- [Example: Requesting an XML Schema on page 211](#)

Requesting an XML Schema for the Configuration Hierarchy

In a NETCONF session with a device running Junos OS, to request an XML Schema-language representation of the entire configuration hierarchy, a client application emits the Junos XML **<get-xnm-information>** tag element and its **<type>** and **<namespace>** child tag elements with the indicated values in an **<rpc>** tag element:

```
<rpc>
  <get-xnm-information>
    <type>xml-schema</type>
    <namespace>junos-configuration</namespace>
  </get-xnm-information>
</rpc>
]]>]]>
```

The NETCONF server encloses the XML schema in **<rpc-reply>** and **<xsd:schema>** tag elements:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <xsd:schema>
    <!-- tag elements for the Junos XML schema -->
  </xsd:schema>
</rpc-reply>
]]>]]>
```

Creating the junos.xsd File

Most of the tag elements defined in the schema returned in the `<xsd:schema>` tag belong to the default namespace for Junos OS configuration elements. However, at least one tag, `<junos:comment>`, belongs to a different namespace:

`http://xml.juniper.net/junos/Junos-version/junos`. By XML convention, a schema describes only one namespace, so schema validators need to import information about any additional namespaces before they can process the schema.

Starting with Junos OS Release 6.4, the `<xsd:import>` tag element is enclosed in the `<xsd:schema>` tag element and references the file `junos.xsd`, which contains the required information about the `junos` namespace. For example, the following `<xsd:import>` tag element specifies the file for Junos OS Release 15.1R1 (and appears on two lines for legibility only):

```
<xsd:import schemaLocation="junos.xsd" \
  namespace="http://xml.juniper.net/junos/15.1R1/junos"/>
```

To enable the schema validator to interpret the `<xsd:import>` tag element, you must manually create a file called `junos.xsd` in the directory where you place the `.xsd` file that contains the complete Junos configuration schema. Include the following text in the file. Do not use line breaks in the list of attributes in the opening `<xsd:schema>` tag. Line breaks appear in the following example for legibility only. For the `Junos-version` variable, substitute the release number of the Junos OS running on the device (for example, 15.1R1 for the first release of Junos OS 15.1).

```
<?xml version="1.0" encoding="us-ascii"?>
<xsd:schema elementFormDefault="qualified" \
  attributeFormDefault="unqualified" \
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" \
  targetNamespace="http://xml.juniper.net/junos/Junos-version/junos">
  <xsd:element name="comment" type="xsd:string"/>
</xsd:schema>
```



NOTE: Schema validators might not be able to process the schema if they cannot locate or open the `junos.xsd` file.

Whenever you change the version of Junos OS running on the device, remember to update the `Junos-version` variable in the `junos.xsd` file to match.

Example: Requesting an XML Schema

The following examples show how to request the Junos OS configuration schema. In the NETCONF server's response, the first `<xsd:element>` statement defines the `<undocumented>` Junos XML tag element, which can be enclosed in most other container tag elements defined in the schema (container tag elements are defined as `<xsd:complexType>`).

The attributes in the opening tags of the NETCONF server's response appear on multiple lines for legibility only. The NETCONF server does not insert newline characters within

tags or tag elements. Also, in actual output the *JUNOS-version* variable is replaced by a value such as 15.1R1 for the initial version of Junos OS Release 15.1.

Client Application NETCONF Server

```
<rpc>
  <get-xnm-information>
    <type>xml-schema</type>
    <namespace>junos-configuration</namespace>
  </get-xnm-information>
</rpc>
]]>]]>

<rpc-reply xmlns="URN" xmlns:junos="URL">
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" \
    elementFormDefault="qualified">
    <xsd:import schemaLocation="junos.xsd" \
      namespace="http://xml.juniper.net/junos/JUNOS-version/junos"/>
    <xsd:element name="undocumented">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="hostname">
      <xsd:simpleContent>
        <xsd:extension base="xsd:string"/>
      </xsd:simpleContent>
    </xsd:complexType>
    .
    .
    .
```

T2114

Another **<xsd:element>** statement near the beginning of the schema defines the Junos XML **<configuration>** tag element. It encloses the **<xsd:element>** statement that defines the **<system>** tag element, which corresponds to the **[edit system]** hierarchy level. The statements corresponding to other hierarchy levels are omitted for brevity.

Client Application	NETCONF Server
---------------------------	-----------------------

```

.
.
.
</xsd:element>
<xsd:element name="configuration">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="undocumented"/>
        <xsd:element ref="comment"/>
        <xsd:element name="system" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="undocumented"/>
                <xsd:element ref="comment"/>
                <!-- child elements of <system> here -->
              </xsd:choice >
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <!-- statements for other hierarchy levels here -->
      </xsd:choice >
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
</rpc-reply>
]]>]]>

```

T2115

- | | |
|------------------------------|--|
| Related Documentation | <ul style="list-style-type: none"> • Requesting Configuration Data Using NETCONF on page 184 • Specifying the Source for Configuration Information Requests Using NETCONF on page 186 • Specifying the Scope of Configuration Information to Return in a NETCONF Response on page 188 |
|------------------------------|--|

CHAPTER 19

NETCONF Perl Client Applications

- [Writing NETCONF Perl Client Applications on page 215](#)
- [Importing Perl Modules and Declaring Constants in NETCONF Perl Client Applications on page 216](#)
- [Connecting to the NETCONF Server in Perl Client Applications on page 217](#)
- [Collecting Parameters Interactively in NETCONF Perl Client Applications on page 219](#)
- [Submitting a Request to the NETCONF Server in Perl Client Applications on page 222](#)
- [Example: Requesting an Inventory of Hardware Components Using a NETCONF Perl Client Application on page 226](#)
- [Example: Changing the Configuration Using a NETCONF Perl Client Application on page 227](#)
- [Parsing and Formatting the Response from the NETCONF Server in Perl Client Applications on page 230](#)
- [Closing the Connection to the NETCONF Server in Perl Client Applications on page 233](#)

Writing NETCONF Perl Client Applications

The following outline lists the basic tasks involved in writing a NETCONF Perl client application that requests operational or configuration information from the NETCONF server or loads configuration information onto a device running Junos OS. Each task provides a link to detailed information about performing the task. The tasks reference the sample scripts included with the NETCONF XML Protocol Perl distribution as examples.

1. [Import Perl Modules and Declare Constants—“Importing Perl Modules and Declaring Constants in NETCONF Perl Client Applications” on page 216](#)
2. [Connect to the NETCONF Server—“Connecting to the NETCONF Server in Perl Client Applications” on page 217 and “Collecting Parameters Interactively in NETCONF Perl Client Applications” on page 219](#)
3. [Submit Requests to the NETCONF Server—“Submitting a Request to the NETCONF Server in Perl Client Applications” on page 222](#)

4. Parse and Format the Response from the NETCONF Server—[“Parsing and Formatting the Response from the NETCONF Server in Perl Client Applications” on page 230](#)
5. Close the Connection to the NETCONF Server—[“Closing the Connection to the NETCONF Server in Perl Client Applications” on page 233](#)

Related Documentation • [Understanding the NETCONF Perl Distribution and Sample Scripts on page 27](#)

[Importing Perl Modules and Declaring Constants in NETCONF Perl Client Applications](#)

When creating a NETCONF Perl client application, include the following statement at the start of the application. This statement imports the functions provided by the **Net::Netconf::Manager** object, which the application uses to connect to the NETCONF server on a device.

```
use Net::Netconf::Manager;
```

Include statements to import other Perl modules as appropriate for your application. For example, several of the sample scripts included in the NETCONF Perl distribution import the following standard Perl modules, which include functions that handle input from the command line:

- **Carp**—Includes functions for user error warnings.
- **Getopt::Std**—Includes functions for reading in keyed options from the command line.
- **Term::ReadKey**—Includes functions for controlling terminal modes, for example suppressing onscreen echo of a typed string such as a password.

If the application uses constants, declare their values at this point. For example, the sample script **diagnose_bgp.pl** includes the following statement to declare a constant for the access method:

```
use constant VALID_ACCESS_METHOD => 'ssh';
```

The **edit_configuration.pl** sample script includes the following statements to declare constants for reporting return codes and the status of the configuration database:

```
use constant REPORT_SUCCESS => 1;
use constant REPORT_FAILURE => 0;
use constant STATE_CONNECTED => 1;
use constant STATE_LOCKED => 2;
use constant STATE_CONFIG_LOADED => 3;
```

Related Documentation • [Writing NETCONF Perl Client Applications on page 215](#)
• [Connecting to the NETCONF Server in Perl Client Applications on page 217](#)

Connecting to the NETCONF Server in Perl Client Applications

The following sections explain how to use the **NET::Netconf::Manager** object in a Perl client application to connect to the NETCONF server on a device running Junos OS:

- [Satisfy Protocol Prerequisites on page 217](#)
- [Group Requests on page 217](#)
- [Obtain and Record Parameters Required by the NET::Netconf::Manager Object on page 217](#)
- [Obtaining Application-Specific Parameters on page 218](#)
- [Establishing the Connection on page 219](#)

Satisfy Protocol Prerequisites

The NETCONF server supports several access protocols. For each connection to the NETCONF server on a device running Junos OS, the application must specify the protocol it is using. Perl client applications can communicate with the NETCONF server via SSH only.

Before your application can run, you must satisfy the prerequisites for SSH. This involves enabling NETCONF on the device by configuring the **set system services netconf ssh** statement.

Group Requests

Establishing a connection to the NETCONF server on a device running Junos OS is one of the more time-intensive and resource-intensive functions performed by an application. If the application sends multiple requests to a device, it makes sense to send all of them within the context of one connection. If your application sends the same requests to multiple devices, you can structure the script to iterate through either the set of devices or the set of requests. Keep in mind, however, that your application can effectively send only one request to one NETCONF server at a time. This is because the **NET::Netconf::Manager** object does not return control to the application until it receives the closing **/rpc-reply** tag that represents the end of the NETCONF server's response to the current request.

Obtain and Record Parameters Required by the NET::Netconf::Manager Object

The **NET::Netconf::Manager** object takes the following required parameters, specified as keys in a Perl hash:

- **access**—The access protocol to use when communicating with the NETCONF server. Before the application runs, satisfy the SSH prerequisites.
- **hostname**—The name of the device to which to connect. For best results, specify either a fully-qualified hostname or an IP address.

- **login**—The username under which to establish the connection to the NETCONF server and issue requests. The username must already exist on the specified device and have the permission bits necessary for making the requests invoked by the application.
- **password**—The password corresponding to the username.

The sample scripts in the NETCONF Perl distribution record the parameters in a Perl hash called **%deviceinfo**, declared as follows:

```
my %deviceinfo = (  
    'access' => $access,  
    'login' => $login,  
    'password' => $password,  
    'hostname' => $hostname,  
);
```

The sample scripts included in the NETCONF Perl client distribution obtain the parameters from options entered on the command line by a user. For more information about collecting parameter values interactively, see [“Collecting Parameters Interactively in NETCONF Perl Client Applications” on page 219](#). Your application can also obtain values for the parameters from a file or database, or you can hardcode one or more of the parameters into the application code if they are constant.

Obtaining Application-Specific Parameters

In addition to the parameters required by the **NET::Netconf::Manager** object, applications might need to define other parameters, such as the name of the file to which to write the data returned by the NETCONF server in response to a request, or the name of the Extensible Stylesheet Transformation Language (XSLT) file to use for transforming the data.

As with the parameters required by the **NET::Netconf::Manager** object, the client application can hardcode the values in the application code, obtain them from a file, or obtain them interactively. The sample scripts obtain values for these parameters from command-line options in the same manner as they obtain the parameters required by the **NET::Netconf::Manager** object. Several examples follow.

The following line enables a debugging trace if the user includes the **-d** command-line option:

```
my $debug_level = $opt{'d'};
```

The following line sets the **\$outputfile** variable to the value specified by the **-o** command-line option. It names the local file to which the NETCONF server's response is written. If the **-o** option is not provided, the variable is set to the empty string.

```
my $outputfile = $opt{'o'} || "";
```

The following code from the `diagnose_bgp.pl` script defines which XSLT file to use to transform the NETCONF server's response. The first line sets the `$xslfile` variable to the value specified by the `-x` command-line option. If the option is not provided, the script uses the `text.xsl` file supplied with the script, which transforms the data to ASCII text. The `if` statement verifies that the specified XSLT file exists; the script terminates if it does not.

```
# Get the xsl file
my $xslfile = $opt{'x'} || "xsl/bgp.xsl";

# Check for the existence of the given file
if (! -f $xslfile) {
    croak "XSL file $xslfile does not exist.";
}
```

Establishing the Connection

After obtaining values for the parameters required for the `NET::Netconf::Manager` object, each sample script records them in the `%deviceinfo` hash.

```
my %deviceinfo = (
    'access' => $access,
    'login' => $login,
    'password' => $password,
    'hostname' => $hostname,
);
```

The script then invokes the NETCONF-specific `new` subroutine to create a `NET::Netconf::Manager` object and establish a connection to the specified routing, switching, or security platform. If the connection attempt fails (as tested by the `ref` operator), the script exits.

```
my $jnx = new Net::Netconf::Manager(%deviceinfo);
unless (ref $jnx) {
    croak "ERROR: $deviceinfo{hostname}: failed to connect.\n";
}
```

Related Documentation

- [Writing NETCONF Perl Client Applications on page 215](#)
- [Importing Perl Modules and Declaring Constants in NETCONF Perl Client Applications on page 216](#)
- [Collecting Parameters Interactively in NETCONF Perl Client Applications on page 219](#)
- [Submitting a Request to the NETCONF Server in Perl Client Applications on page 222](#)
- [Closing the Connection to the NETCONF Server in Perl Client Applications on page 233](#)

Collecting Parameters Interactively in NETCONF Perl Client Applications

In a NETCONF Perl client application, a script can interactively obtain the parameters required by the `NET::Netconf::Manager` object from command-line.

The NETCONF Perl distribution includes several sample Perl scripts to perform various functions on devices running Junos OS. Each sample script obtains the parameters required by the `NET::Netconf::Manager` object from command-line options provided by

the user who invokes the script. The script records the options in a Perl hash called `%opt`, using the `getopts` function defined in the `Getopt::Std` Perl module to read the options from the command line. (Scripts used in production environments probably do not obtain parameters interactively, so this section is important mostly for understanding the sample scripts.)

In the following example from the `get_chassis_inventory.pl` sample script, the first parameter to the `getopts` function defines the acceptable options, which vary depending on the application. A colon after the option letter indicates that it takes an argument.

The second parameter, `\%opt`, specifies that the values are recorded in the `%opt` hash. If the user does not provide at least one option, provides an invalid option, or provides the `-h` option, the script invokes the `output_usage` subroutine, which prints a usage message to the screen.

```
my %opt;
getopts('!p:d:x:f:m:o:h', \%opt) || output_usage();
output_usage() if $opt{'h'};
```

The following code defines the `output_usage` subroutine for the `get_chassis_inventory.pl` sample script. The contents of the `my $usage` definition and the `Where` and `Options` sections are specific to the script, and differ for each application.

```
sub output_usage
{
    my $usage = "Usage: $0 [options] <target>
```

Where:

<target> The hostname of the target device.

Options:

```
-l <login> A login name accepted by the target device.
-p <password> The password for the login name.
-m <access> Access method. The only supported method is 'ssh'.
-x <format> The name of the XSL file to display the response.
             Default: xsl/chassis_inventory.xsl
-f <xmlfile> The name of the XML file to print server response to.
             Default: xsl/chassis_inventory.xml
-o <filename> output is written to this file instead of standard output.
-d <level> Debug level [1-6]\n\n";
```

```
    croak $usage;
}
```

The `get_chassis_inventory.pl` script includes the following code to obtain values from the command line for the parameters required by the `NET::Netconf::Manager` object. A detailed discussion of the various functional units follows the complete code sample.

```
# Get the hostname
my $hostname = shift || output_usage();

# Get the access method, can be ssh only
my $access = $opt{'m'} || 'ssh';
use constant VALID_ACCESS_METHOD => 'ssh';
```



```

output_usage() unless (VALID_ACCESS_METHOD =~ /$access/);

# Check for login name. If not provided, prompt for it
my $login = "";
if ($opt{'l'}) {
    $login = $opt{'l'};
} else {
    print STDERR "login: ";
    $login = ReadLine 0;
    chomp $login;
}

# Check for password. If not provided, prompt for it
my $password = "";
if ($opt{'p'}) {
    $password = $opt{'p'};
} else {
    print STDERR "password: ";
    ReadMode 'noecho';
    $password = ReadLine 0;
    chomp $password;
    ReadMode 'normal';
    print STDERR "\n";
}

```

In the first line of the preceding code sample, the script uses the Perl **shift** function to read the hostname from the end of the command line. If the hostname is missing, the script invokes the **output_usage** subroutine to print the usage message, which specifies that a hostname is required.

```
my $hostname = shift || output_usage();
```

The script next determines which access protocol to use, setting the **\$access** variable to the value of the **-m** command-line option. If the specified value does not match the only valid value defined by the **VALID_ACCESSSES** constant, the script invokes the **output_usage** subroutine to print the usage message.

```

my $access = $opt{'m'} || 'ssh';
use constant VALID_ACCESS_METHOD => 'ssh';
output_usage() unless (VALID_ACCESS_METHOD =~ /$access/);

```

The script then determines the username, setting the **\$login** variable to the value of the **-l** command-line option. If the option is not provided, the script prompts for it and uses the **ReadLine** function (defined in the standard Perl **Term::ReadKey** module) to read it from the command line.

```

my $login = "";
if ($opt{'l'}) {
    $login = $opt{'l'};
} else {
    print STDERR "login: ";
    $login = ReadLine 0;
    chomp $login;
}

```

The script finally determines the password for the username, setting the `$password` variable to the value of the `-p` command-line option. If the option is not provided, the script prompts for it. It uses the `ReadMode` function (defined in the standard Perl `Term::ReadKey` module) twice: first to prevent the password from echoing visibly on the screen and then to return the shell to normal (echo) mode after it reads the password.

```
my $password = "";
if ($opt{'p'}) {
    $password = $opt{'p'};
} else {
    print STDERR "password: ";
    ReadMode 'noecho';
    $password = ReadLine 0;
    chomp $password;
    ReadMode 'normal';
    print STDERR "\n";
}
```

Related Documentation • [Writing NETCONF Perl Client Applications on page 215](#)

Submitting a Request to the NETCONF Server in Perl Client Applications

In a NETCONF Perl client application, after establishing a connection to a NETCONF server, the client application can submit one or more requests by invoking the Perl methods that are supported in the version of the NETCONF XML protocol and Junos XML API used by the application.

- Each version of software supports a set of methods that correspond to CLI operational mode commands. For a list of the operational methods supported in the current version, see the files stored in the `lib/Net/Netconf/Plugins/Plugin/release` directory of the NETCONF Perl distribution (*release* is the Junos OS version code, such as 6.1R1 for the initial version of Junos OS Release 6.1). The files have names in the format `package_methods.pl`, where *package* is a software package.
- The set of methods that correspond to operations on configuration objects is defined in the `lib/Net/Netconf/Plugins.pm` file in the NETCONF distribution.

See the following sections for more information:

- [Providing Method Options or Attributes on page 222](#)
- [Submitting a Request on page 224](#)

Providing Method Options or Attributes

Many Perl methods have one or more options or attributes. The following list describes the notation used to define a method's options in the `lib/Net/Netconf/Plugins.pm` and `lib/Net/Netconf/release/package_methods.pl` files, and the notation that an application uses when invoking the method:

- A method without options is defined as `$NO_ARGS`, as in the following entry for the `get_autoinstallation_status_information` method:

```
## Method : get-autoinstallation-status-information
## Returns: autoinstallation-status-information
## Command: "show system autoinstallation status"
get_autoinstallation_status_information => $NO_ARGS,
```

To invoke a method without options, follow the method name with an empty set of parentheses as in the following example:

```
$jnx->get_autoinstallation_status_information( );
```

- A fixed-form option is defined as type **\$TOGGLE**. In the following example, the **get_software_information** method takes two fixed-form options, **brief** and **detail**:

```
## Method : <get-ancp-neighbor-information>
## Returns: <ancp-neighbor-information>
## Command: "show ancp neighbor"
get_ancp_neighbor_information => {
    brief => $TOGGLE,
    detail => $TOGGLE,
}
```

To include a fixed-form option when invoking a method, set it to the value **1** (one) as in the following example:

```
$jnx->get-ancp-neighbor-information(brief => 1);
```

- An option with a variable value is defined as type **\$STRING**. In the following example, the **get_cos_drop_profile_information** method takes the **profile_name** argument:

```
## Method : <get-passive-monitoring-usage-information>
## Returns: <passive-monitoring-usage-information>
## Command: "show passive-monitoring usage"
get_passive_monitoring_usage_information => {
    interface_name => $STRING,
}
```

To include a variable value when invoking a method, enclose the value in single quotes as in the following example:

```
$jnx->get_cos_drop_profile_information(profile_name => 'user-drop-profile');
```

- A set of configuration statements or corresponding tag elements is defined as type **\$DOM**. In the following example, the **get_config** method takes a set of configuration statements (along with two attributes):

```
'get_config' => {
    'source' => $DOM_STRING,
    'source_url' => $URL_STRING,
    'filter' => $DOM
},
```

A DOM object is XML code:

```
my $xml_string = "
<filter type=\"subtree\">
<configuration>
  <protocols>
    <bgp></bgp>
  </protocols>
</configuration>
```

```
</filter>
";

my %queryargs = (
    'source' => "running",
    'filter' => $xml_string,
);
```

This generates an RPC request:

```
<rpc message-id='1'> <get-config> <source> <running/> </source>
<filter type="subtree">
<configuration>
  <protocols>
    <bgp></bgp>
  </protocols>
</configuration>
</filter>
</get-config></rpc>
```

A method can have a combination of fixed-form options, options with variable values, attributes, and a set of configuration statements. For example, the **get_forwarding_table_information** method has four fixed-form options and five options with variable values:

```
## Method : get-forwarding-table-information
## Returns: forwarding-table-information
## Command: "show route forwarding-table"
get_forwarding_table_information => {
    detail => $TOGGLE,
    extensive => $TOGGLE,
    multicast => $TOGGLE,
    family => $STRING,
    vpn => $STRING,
    summary => $TOGGLE,
    matching => $STRING,
    destination => $STRING,
    label => $STRING,
},
```

Submitting a Request

The following code is the recommended way to send a request to the NETCONF server and shows how to handle error conditions. The `$jnx` variable is defined to be a **NET::Netconf::Manager** object.

```
my $res; # Netconf server response

# connect to the Netconf server
my $jnx = new Net::Netconf::Manager(%deviceinfo);
unless (ref $jnx) {
    croak "ERROR: $deviceinfo{hostname}: failed to connect.\n";
}

# Lock the configuration database before making any changes
print "Locking configuration database ... \n";
```

```

my %queryargs = ( 'target' => 'candidate' );
$res = $jnx->lock_config(%queryargs);

# See if you got an error
if ($jnx->has_error) {
    print "ERROR: in processing request \n $jnx->{'request'} \n";
    graceful_shutdown($jnx, STATE_CONNECTED, REPORT_FAILURE);
}

# Load the configuration from the given XML file
print "Loading configuration from $xmlfile \n";
if (! -f $xmlfile) {
    print "ERROR: Cannot load configuration in $xmlfile\n";
    graceful_shutdown($jnx, STATE_LOCKED, REPORT_FAILURE);
}

# Read in the XML file
my $config = read_xml_file($xmlfile);
print "\n\n$config \n\n";

%queryargs = (
    'target' => 'candidate',
    'config' => $config
);
$res = $jnx->edit_config(%queryargs);

# See if you got an error
if ($jnx->has_error) {
    print "ERROR: in processing request \n $jnx->{'request'} \n";
    # Get the error
    my $error = $jnx->get_first_error();
    get_error_info(%$error);
    # Disconnect
    graceful_shutdown($jnx, STATE_LOCKED, REPORT_FAILURE);
}

# Commit the changes
print "Committing the edit-config changes ... \n";
$jnx->commit();
if ($jnx->has_error) {
    print "ERROR: Failed to commit the configuration.\n";
    graceful_shutdown($jnx, STATE_CONFIG_LOADED, REPORT_FAILURE);
}

# Unlock the configuration database and
# disconnect from the Netconf server
print "Disconnecting from the Netconf server ... \n";
graceful_shutdown($jnx, STATE_LOCKED, REPORT_SUCCESS);

```

Related Documentation

- [Writing NETCONF Perl Client Applications on page 215](#)
- [Example: Requesting an Inventory of Hardware Components Using a NETCONF Perl Client Application on page 226](#)
- [Example: Changing the Configuration Using a NETCONF Perl Client Application on page 227](#)

- [Parsing and Formatting the Response from the NETCONF Server in Perl Client Applications on page 230](#)

Example: Requesting an Inventory of Hardware Components Using a NETCONF Perl Client Application

The NETCONF Perl distribution includes several sample Perl scripts to perform various functions on devices running Junos OS. The `get_chassis_inventory.pl` script retrieves and displays a detailed inventory of the hardware components installed in a routing, switching, or security platform. It is equivalent to issuing the `show chassis hardware detail` operational mode command in the Junos OS command-line interface (CLI). This topic describes the portion of the script that executes the query.

After establishing a connection to the NETCONF server, the script sends the `get_chassis_inventory` request and includes the `detail` argument.

```
my $query = "get_chassis_inventory";
my %queryargs = ( 'detail' => 1 );
```

The script sends the query and assigns the results to the `$res` variable. It performs two tests on the results, and prints an error message if it cannot send the request or if errors occurred when executing it. If no errors occur, the script uses XSLT to transform the results.

```
# send the command and get the server response
my $res = $jnx->$query(%queryargs);
print "Server request: \n $jnx->{'request'}\n Server response: \n $jnx->{'server_response'}\n";

# print the server response into xmlfile
print_response($xmlfile, $jnx->{'server_response'});

# See if you got an error
if ($jnx->has_error) {
    croak "ERROR: in processing request \n $jnx->{'request'} \n";
} else {
    # Transform the server response using XSL file
    my $res = new Net::Netconf::Transform();
    print "Transforming ... \n";
    my $nm = $res->translateXSLtoRelease('xmlns:lc', $xslfile,
                                        "$xslfile.tmp",
                                        $xmlfile);

    if ($nm) {
        format_by_xslt($nm, $xmlfile, );
    } else {
        print STDERR "ERROR: Invalid XSL File $xslfile\n";
    }
}

# Disconnect from the Netconf server
$jnx->disconnect();
```

Related
Documentation

- [Writing NETCONF Perl Client Applications on page 215](#)

- [Submitting a Request to the NETCONF Server in Perl Client Applications on page 222](#)

Example: Changing the Configuration Using a NETCONF Perl Client Application

The NETCONF Perl distribution includes several sample Perl scripts to perform various functions on devices running Junos OS. The `edit_configuration.pl` script locks, modifies, uploads, and commits the configuration on a device. It uses the basic structure for sending requests but also defines a `graceful_shutdown` subroutine that handles errors. The following sections describe the different functions that the script performs:

- [Handling Error Conditions on page 227](#)
- [Locking the Configuration on page 228](#)
- [Reading In the Configuration Data on page 228](#)
- [Editing the Configuration Data on page 229](#)
- [Committing the Configuration on page 230](#)

Handling Error Conditions

The `graceful_shutdown` subroutine in the `edit_configuration.pl` script handles errors encountered in the NETCONF session. It employs the following additional constants:

```
# query execution status constants
use constant REPORT_SUCCESS => 1;
use constant REPORT_FAILURE => 0;
use constant STATE_CONNECTED => 1;
use constant STATE_LOCKED => 2;
use constant STATE_CONFIG_LOADED => 3;
```

The first two `if` statements in the subroutine refer to the `STATE_CONFIG_LOADED` and `STATE_LOCKED` conditions, which apply specifically to loading a configuration in the `edit_configuration.pl` script.

```
sub graceful_shutdown
{
    my ($jnx, $state, $success) = @_;
    if ($state >= STATE_CONFIG_LOADED) {
        # We have already done an <edit-config> operation
        # - Discard the changes
        print "Discarding the changes made ...\n";
        $jnx->discard_changes();
        if ($jnx->has_error) {
            print "Unable to discard <edit-config> changes\n";
        }
    }

    if ($state >= STATE_LOCKED) {
        # Unlock the configuration database
        $jnx->unlock_config();
        if ($jnx->has_error) {
            print "Unable to unlock the candidate configuration\n";
        }
    }
}
```

```
if ($state >= STATE_CONNECTED) {
    # Disconnect from the Netconf server
    $jnx->disconnect();
}

if ($success) {
    print "REQUEST succeeded !!\n";
} else {
    print "REQUEST failed !!\n";
}

exit;
}
```

Locking the Configuration

The main section of the `edit_configuration.pl` script begins by establishing a connection to a NETCONF server. It then invokes the `lock_configuration` method to lock the configuration database. If an error occurs, the script invokes the `graceful_shutdown` subroutine described in [“Handling Error Conditions” on page 227](#).

```
print "Locking configuration database ...\n";
my %queryargs = ( 'target' => 'candidate' );
$res = $jnx->lock_config(%queryargs);
# See if you got an error
if ($jnx->has_error) {
    print "ERROR: in processing request \n $jnx->{'request'} \n";
    graceful_shutdown($jnx, STATE_CONNECTED, REPORT_FAILURE);
}
```

Reading In the Configuration Data

In the following code sample, the `edit_configuration.pl` script reads in and parses a file that contains Junos XML configuration tag elements or ASCII-formatted statements. A detailed discussion of the functional subsections follows the complete code sample.

```
# Load the configuration from the given XML file
print "Loading configuration from $xmlfile \n";
if (! -f $xmlfile) {
    print "ERROR: Cannot load configuration in $xmlfile\n";
    graceful_shutdown($jnx, STATE_LOCKED, REPORT_FAILURE);
}

# Read in the XML file
my $config = read_xml_file($xmlfile);
print "\n\n$config \n\n";

%queryargs = (
    'target' => 'candidate'
);

# If we are in text mode, use config-text arg with wrapped
# configuration-text, otherwise use config arg with raw XML
if ($opt{t}) {
    $queryargs{'config-text'} = '<configuration text> . $config . </configuration-text>';
}
```



```

} else {
    $queryargs{'config'} = $config;

```

The first subsection of the preceding code sample verifies the existence of the file containing configuration data. The name of the file was previously obtained from the command line and assigned to the `$xmlfile` variable. If the file does not exist, the script invokes the `graceful_shutdown` subroutine.

```

print "Loading configuration from $xmlfile \n";
if (! -f $xmlfile) {
    print "ERROR: Cannot load configuration in $xmlfile\n";
    graceful_shutdown($jnx, STATE_LOCKED, REPORT_FAILURE);
}

```

The script then invokes the `read_xml_file` subroutine, which opens the file for reading and assigns its contents to the `$config` variable. The `queryargs` key `target` is set to the value `candidate`. When the script calls the `edit_configuration` method, the candidate configuration is edited.

```

# Read in the XML file
my $config = read_xml_file($xmlfile);
print "\n\n$config \n\n";

%queryargs = (
    'target' => 'candidate'
);

```

If the `-t` command-line option was included when the `edit_configuration.pl` script was invoked, the file referenced by the `$xmlfile` variable should contain ASCII-formatted configuration statements like those returned by the CLI configuration-mode `show` command. If the configuration statements are in ASCII-formatted text, the script encloses the configuration stored in the `$config` variable within the `configuration-text` tag element and stores the result in the value associated with the `queryargs` hash key `config-text`.

If the `-t` command-line option was not included when the `edit_configuration.pl` script was invoked, the file referenced by the `$xmlfile` variable contains Junos XML configuration tag elements. In this case, the script stores just the `$config` variable as the value associated with the `queryargs` hash key `config`.

```

if ($opt{t}) {
    $queryargs{'config-text'} = '<configuration text> . $config . </configuration-text>';
} else {
    $queryargs{'config'} = $config;
}

```

Editing the Configuration Data

The script invokes the `edit_config` method to load the configuration changes onto the device. It invokes the `graceful_shutdown` subroutine if the response from the NETCONF server has errors.

```

$res = $jnx->edit_config(%queryargs);

# See if you got an error

```

```
if ($jnx->has_error) {  
    print "ERROR: in processing request \n $jnx->{'request'} \n";  
    # Get the error  
    my $error = $jnx->get_first_error();  
    get_error_info(%$error);  
    # Disconnect  
    graceful_shutdown($jnx, STATE_LOCKED, REPORT_FAILURE);  
}
```

Committing the Configuration

If there are no errors up to this point, the script invokes the **commit** method to commit the configuration on the device and make it the active configuration.

```
# Commit the changes  
print "Committing the <edit-config> changes ... \n";  
$jnx->commit();  
if ($jnx->has_error) {  
    print "ERROR: Failed to commit the configuration. \n";  
    graceful_shutdown($jnx, STATE_CONFIG_LOADED, REPORT_FAILURE);  
}
```

- Related Documentation**
- [Writing NETCONF Perl Client Applications on page 215](#)
 - [Submitting a Request to the NETCONF Server in Perl Client Applications on page 222](#)

Parsing and Formatting the Response from the NETCONF Server in Perl Client Applications

In a NETCONF Perl client application, as the last step in sending a request, the application verifies that there are no errors with the response from the NETCONF server. It can then write the response to a file, to the screen, or both. If the response is for an operational query, the application usually uses XSLT to transform the output into a more readable format, such as HTML or formatted ASCII. If the response consists of configuration data, the application can store it as XML (the Junos XML tag elements generated by default from the NETCONF server) or transform it into formatted ASCII text.

The following code sample from the **diagnose_bgp.pl** script included in the NETCONF Perl distribution uses XSLT to transform an operational response from the NETCONF server into a more readable format. A detailed discussion of the functional subsections follows the complete code sample.

```
# Get the output file  
my $outputfile = $opt{'o'} || "";  
  
# Get the xsl file  
my $xslfile = $opt{'x'} || "xsl/bgp.xsl";  
  
# Check for the existence of the given file  
if (! -f $xslfile) {  
    croak "XSL file $xslfile does not exist.";  
}  
  
# Get the xmlfile  
my $xmlfile = $opt{'f'} || "xsl/bgp.xml";
```

```

# send the command and get the server response
my $res = $jnx->$query();

# print the server response into xmlfile
print_response($xmlfile, $jnx->{'server_response'});

# See if you got an error
if ($jnx->has_error) {
    croak "ERROR: in processing request \n $jnx->{'request'} \n";
} else {
    # Transform the server response using XSL file
    my $res = new Net::Netconf::Transform();
    print "Transforming ...\n";
    my $nm = $res->translateXSLtoRelease('xmlns:lc', $xslfile,
                                        "$xslfile.tmp",
                                        $xmlfile);

    if ($nm) {
        format_by_xslt($nm, $xmlfile, );
    } else {
        print STDERR "ERROR: Invalid XSL File $xslfile\n";
    }
}

```

The first line of the preceding code sample illustrates how the scripts read the `-o` option from the command line to obtain the name of the file into which to write the results of the XSLT transformation:

```
my $outputfile = $opt{'o'} || "";
```

From the `-x` command-line option, the scripts obtain the name of the XSLT file to use, setting a default value if the option is not provided. The scripts exit if the specified file does not exist. The following example is from the **diagnose_bgp.pl** script:

```

my $xslfile = $opt{'x'} || "xsl/bgp.xsl";
if (! -f $xslfile) {
    croak "XSL file $xslfile does not exist.";
}

```

For examples of XSLT files, see the following directories in the NETCONF Perl distribution:

- The **examples/diagnose_bgp/xsl** directory contains an XSLT file for the **diagnose_bgp.pl** script.
- The **examples/get_chassis_inventory/xsl** directory contains XSLT files for the **get_chassis_inventory.pl** script.

The actual parsing operation invokes the **translateXSLtoRelease** function (defined in the **Net::Netconf::Transform** module) to alter one of the namespace definitions in the XSLT file.

```

my $res = new Net::Netconf::Transform();
print "Transforming ...\n";
my $nm = $res->translateXSLtoRelease('xmlns:lc', $xslfile,
                                    "$xslfile.tmp",
                                    $xmlfile);

if ($nm) {

```

```
    format_by_xslt($nm, $xmlfile, );  
  } else {  
    print STDERR "ERROR: Invalid XSL File $xslfile\n";  
  }  
}
```

This is necessary because the XSLT 1.0 specification requires that every XSLT file define a specific value for each default namespace used in the data being transformed. The **xmlns** attribute in a NETCONF operational response tag element includes a code representing the Junos OS version, such as 10.3R1 for the initial version of Junos OS Release 10.3. Because the same XSLT file can be applied to operational response tag elements from devices running different versions of the Junos OS, the XSLT file cannot predefine an **xmlns** namespace value that matches all versions. The **translateXSLtoRelease** function alters the namespace definition in the XSLT file identified by the **\$xslfile** variable to match the value in the NETCONF server's response. It assigns the resulting XSLT file to the **\$nm** variable.

After verifying that the **translateXSLtoRelease** function succeeded, the script invokes the **format_by_xslt** function, which builds a command string and assigns it to the **\$command** variable. The first part of the command string invokes the **xsltproc** command and specifies the names of the XSLT and configuration data files (**\$xslfile** and **\$xmlfile**):

```
sub format_by_xslt  
{  
  my ($xslfile, $xmlfile, $outfile) = @_;  
  
  print "Transforming $xmlfile with $xslfile...\n" if $outfile;  
  my $command = "xsltproc $xslfile $xmlfile";  
  $command .= "> $outfile" if $outfile;  
  system($command);  
  print "Done\n" if $outfile;  
  print "See $outfile\n" if $outfile;  
}
```

If the **\$outfile** variable is defined (the file for storing the result of the XSLT transformation exists), the script appends a string to the **\$command** variable to write the results of the **xsltproc** command to the file. (If the file does not exist, the script writes the results to standard out [stdout].) The script then invokes the **system** function to execute the command string and prints status messages to stdout.

If the **translateXSLtoRelease** function fails (the **if (\$nm)** expression evaluates to "false"), the script prints an error:

```
if ($nm) {  
  format_by_xslt($nm, $xmlfile, );  
} else {  
  print STDERR "ERROR: Invalid XSL File $xslfile\n";  
}
```

- Related Documentation**
- [Writing NETCONF Perl Client Applications on page 215](#)
 - [Submitting a Request to the NETCONF Server in Perl Client Applications on page 222](#)

Closing the Connection to the NETCONF Server in Perl Client Applications

In NETCONF Perl client applications, you can end the NETCONF session and close the connection to the device by invoking the **disconnect** method.

Several of the sample scripts included in the NETCONF Perl client distribution invoke the **disconnect** method in standalone statements. For example:

```
$jnx->disconnect();
```

The **edit_configuration.pl** sample script invokes the **graceful_shutdown** method, which takes the appropriate actions with regard to the configuration database and then invokes the **disconnect** method.

```
graceful_shutdown($jnx, $xmlfile, STATE_LOCKED, REPORT_SUCCESS);
```

Related Documentation

- [Writing NETCONF Perl Client Applications on page 215](#)
- [Connecting to the NETCONF Server in Perl Client Applications on page 217](#)

CHAPTER 20

YANG

- [Using Juniper Networks YANG Modules on page 235](#)
- [show system schema](#)

Using Juniper Networks YANG Modules

Juniper Networks provides YANG modules that define the Junos OS configuration hierarchy and Junos OS YANG extensions. You can use existing YANG-based tools or develop custom network management applications that utilize the YANG modules to generate vendor-specific configuration data for different devices or validate data before uploading it to a device.

[Table 5 on page 235](#) summarizes the Juniper Networks YANG modules and their associated namespace URI and prefix. The **configuration** module imports the **junos-extension** module, which contains definitions for Junos OS YANG extensions.

Table 5: Juniper Networks YANG Modules

Module	Description	Namespace URI	Prefix
configuration	Defines the Junos OS configuration hierarchy.	http://yang.juniper.net/yang/1.1/jc	jc
junos-extension	Contains definitions for Junos OS YANG extensions including the must and must-message keywords, which identify configuration hierarchy constraints that cannot be mapped directly to YANG's must statement.	http://yang.juniper.net/yang/1.1/je	junos

The following sections detail how to obtain Juniper Networks YANG modules and how to import them into another module:

1. [Obtaining Juniper Networks YANG Modules on page 236](#)
2. [Importing Juniper Networks YANG Modules on page 236](#)

Obtaining Juniper Networks YANG Modules

You can download Juniper Networks YANG modules from the website, or you can generate the modules on the device running Junos OS. When downloaded from the website, the YANG module for the Junos OS configuration hierarchy defines the complete hierarchy for all devices running that Junos OS release. When generated on the local device, the YANG module defines the hierarchy specific to that device.

To download the generic Juniper Networks YANG modules:

1. Access the support page at <http://www.juniper.net/support/downloads/junos.html>.
2. Select your product.
3. Select the appropriate software version.
4. Select the Software tab.
5. In the Tools section, click the link for the YANG modules.

To generate device-specific YANG modules:

1. Log in to the device running Junos OS.
2. Execute the **show system schema** operational mode command, and specify the module name, the YANG format, and optionally, a file for the output.

```
user@host> show system schema module module-name format yang output-file-name  
                               filename
```

For example, the following command generates a YANG module that defines the device-specific configuration hierarchy and saves it to a file:

```
user@host> show system schema module configuration format yang output-file-name  
/var/tmp/config.yang
```

Importing Juniper Networks YANG Modules

You can use YANG-based tools to leverage the Juniper Networks YANG modules. If you are developing custom YANG modules, you can reference definitions in the Juniper Networks YANG modules by importing the modules into your custom module.

To import Juniper Networks YANG modules into an existing module:

1. Include the import statement, specify the module name, and assign the prefix to use with the definitions from the imported module.

```
module test-system {  
  namespace "http://test.example.com/system";  
  prefix "test";  
  
  import configuration {  
    prefix "jc";  
  }  
  import junos-extension {  
    prefix "junos";  
  }  
}
```



```
...  
}
```

2. Reference definitions in the module by using the locally defined prefix, a colon, and the node identifier or keyword.

For example, to reference the **interface** node, use **jc:interface**.

**Related
Documentation**

- [Understanding YANG on Devices Running Junos OS on page 29](#)
- [YANG Modules Overview on page 30](#)
- [show system schema on page 238](#)

show system schema

Syntax	show system schema module (configuration junos-extension) <format <i>format</i> > <output-file-name <i>filename</i> >
Release Information	Command introduced in Junos OS Release 14.2. Option module junos-extension introduced in Junos OS Release 15.1.
Description	<p>Display the device-specific Junos OS schema in the specified format. If you do not specify a format, by default, the device displays the output as a YANG module.</p> <p>The Junos XML RPC for this operational mode command is:</p> <pre><rpc> <get-schema> <format><i>format</i></format> <identifier>(configuration junos-extension)</identifier> <output-file-name><i>filename</i></output-file-name> </get-schema> </rpc></pre>
Options	<p>format <i>format</i>—(Optional) Data modeling language of the schema. Specify yang to display the schema as a YANG module.</p> <p>Default: yang</p> <p>module (configuration junos-extension)—Module to display. Specify configuration to view the schema for Junos OS configuration data or junos-extension to view Junos OS YANG extensions.</p> <p>output-file-name <i>filename</i>—(Optional) File to which the output is written. If you do not specify an absolute path, the device places the file in the current working directory, which defaults to the user's home directory in /var/home. If you omit this option, the output is sent to standard output.</p>
Required Privilege Level	view
Related Documentation	<ul style="list-style-type: none">• Understanding YANG on Devices Running Junos OS on page 29• YANG Modules Overview on page 30• Using Juniper Networks YANG Modules on page 235

PART 5

Index

- [Index on page 241](#)

Index

Symbols

#, comments in configuration statements.....	xvii
(), in syntax descriptions.....	xvii
< >, in syntax descriptions.....	xvii
[], in configuration statements.....	xvii
]]>]]> character sequence (NETCONF).....	137
usage guidelines.....	24
{ }, in configuration statements.....	xvii
(pipe), in syntax descriptions.....	xvii

A

abort tag (Junos XML protocol).....	143
abort-acknowledgement tag (Junos XML protocol).....	143
access	
protocols for NETCONF.....	51
action attribute (Junos XML protocol)	
load-configuration tag	156
at-time tag (Junos XML protocol).....	145
attributes	
in the rpc tag echoed in rpc-reply.....	65
Junos XML tags See Index of Tag Elements and Attributes for list See names of individual attributes for usage guidelines	
NETCONF tags See Index of Tag Elements and Attributes for list See names of individual attributes for usage guidelines	
authentication	
NETCONF.....	60

B

bad-element tag (NETCONF).....	138
usage guidelines.....	70
braces, in configuration statements.....	xvii
brackets	
angle, in syntax descriptions.....	xvii
square, in configuration statements.....	xvii

C

candidate tag (NETCONF).....	141
replacing entire configuration	
usage guidelines.....	94
requesting configuration information	
usage guidelines.....	186
requesting information.....	133
unlocking configuration.....	72
validating configuration.....	135
usage guidelines.....	113
capabilities tag (NETCONF).....	138
usage guidelines.....	61
capability tag (NETCONF).....	138
usage guidelines.....	61
changed attribute (Junos XML protocol)	
get-configuration tag.....	152
check tag (Junos XML protocol).....	145
checksum attribute	
checksum-information tag.....	144
checksum-information (Junos XML protocol).....	144
child tags (XML) See tags (XML)	
CLI configuration data	
in configuration statements.....	88
client applications	
NETCONF See NETCONF client	
applications.....	27
close-configuration tag (Junos XML protocol).....	144
close-session tag (NETCONF).....	127
usage guidelines.....	75
command output	
JSON format, displaying.....	152, 175, 177
RPC, displaying.....	7
commands	
mapping options to Junos XML tags	
fixed-form.....	15
variable-form.....	15
comments	
about configuration, Junos XML mapping.....	20
NETCONF and XML.....	12
comments, in configuration statements.....	xvii
commit tag (NETCONF).....	127
usage guidelines	
confirmed commit.....	114
regular commit.....	114
commit-at tag (Junos XML protocol)	
database-status tag.....	150
commit-check-success tag (Junos XML protocol).....	162

commit-configuration tag (Junos XML protocol).....	145
commit-results tag (Junos XML protocol).....	149
commit-scripts attribute (Junos XML protocol)	
get-configuration tag.....	152
commit-success tag (Junos XML protocol).....	162
compare attribute (Junos XML protocol)	
get-configuration tag.....	152
compare tag (Junos XML).....	206
compatibility	
between NETCONF server and application.....	63
computation-method attribute	
checksum-information tag.....	144
config tag (NETCONF).....	129
usage guidelines.....	87
configuration	
adding comments	
Junos XML.....	20
committing	
confirmation required (NETCONF).....	114
immediately (NETCONF).....	114
comparing with previous	
NETCONF.....	206
creating	
element only if new (NETCONF).....	99
deleting	
hierarchy level (NETCONF).....	102
multiple values from leaf	
(NETCONF).....	104
object (NETCONF).....	102
overview (NETCONF).....	101
single option (NETCONF).....	103
deleting candidate.....	96
discarding changes	
NETCONF.....	95
displaying	
candidate or committed	
(NETCONF).....	186
entire (NETCONF).....	188
hierarchy level (NETCONF).....	189
identifiers (NETCONF).....	194
multiple elements at once	
(NETCONF).....	203
objects of specific type (NETCONF).....	191
overview (NETCONF).....	183, 184
rescue (NETCONF).....	208
rollback (NETCONF).....	204
single object (NETCONF).....	197
specific children of object	
(NETCONF).....	199
XML schema for.....	210
editing	
individual elements.....	96
loading	
as a data stream (NETCONF).....	87
as data in a file (NETCONF).....	85
default mode for NETCONF.....	90
locking (NETCONF).....	72
merge data mode.....	91
merging current and new (NETCONF).....	98
NETCONF.....	24
no-change data mode.....	92
replace data mode.....	91
replacing	
entire (NETCONF).....	94
patterns (Junos XML protocol).....	108
single element (NETCONF).....	106
rescue	
displaying (NETCONF).....	208
reusing response tags.....	21
rolling back to previous	
NETCONF.....	95
statements See configuration statements	
unlocking (NETCONF).....	73
verifying (NETCONF).....	113
configuration data	
data files.....	85
data format	
CLI configuration data.....	88
Junos XML.....	88
streaming data.....	85
configuration statements	
adding comments about	
Junos XML.....	20
deleting (NETCONF).....	101
mapping to Junos XML tags	
comments.....	20
hierarchy level or container tag.....	16
identifiers.....	17
keywords.....	17
leaf statements.....	18
multiple options on one line.....	19
multiple values for an option.....	19
configuration tag (Junos XML).....	16
configuration tag (NETCONF).....	129
requesting information.....	132, 133

-
- configuration-information tag (Junos XML)
 - comparing configurations.....206
 - displaying configuration.....204
 - configuration-output tag (Junos XML)
 - comparing configurations.....206
 - displaying configuration.....204
 - configure-exclusive tag (Junos XML protocol)
 - database-status tag.....150
 - confirm-timeout (NETCONF).....127
 - confirm-timeout tag (Junos XML protocol).....145
 - confirmed tag (Junos XML protocol).....145
 - confirmed tag (NETCONF).....127
 - usage guidelines.....114
 - confirmed-timeout tag (NETCONF)
 - usage guidelines.....114
 - connection-limit statement.....118
 - conventions
 - for client to comply with.....10
 - text and syntax.....xvi
 - copy-config tag (NETCONF).....128
 - usage guidelines.....83, 94
 - create (NETCONF 'operation' attribute)
 - usage guidelines.....99
 - curly braces, in configuration statements.....xvii
 - customer support.....xviii
 - contacting JTAC.....xviii
 - D**
 - daemon tag (Junos XML protocol).....161
 - data files
 - configuration data.....85
 - referencing configuration data.....85
 - data tag (NETCONF).....137
 - usage guidelines.....183, 184
 - database attribute (Junos XML protocol)
 - get-configuration tag.....152
 - database-status tag (Junos XML protocol).....150
 - database-status-information tag (Junos XML protocol).....151
 - dedicated port
 - NETCONF over SSH.....55
 - default mode for NETCONF configuration
 - changes.....90
 - default-operation tag (NETCONF).....129
 - usage guidelines
 - deleting configuration.....101
 - general.....90
 - replacing configuration.....94
 - delete (NETCONF 'operation' attribute)
 - usage guidelines.....101
 - delete-config tag.....129
 - discard tag (NETCONF)
 - usage guidelines.....95
 - discard-changes tag.....129
 - discard-changes tag (NETCONF)
 - changing configuration.....83
 - display json filter.....152, 177
 - display xml command
 - usage guidelines.....7
 - display xml rpc command
 - usage guidelines.....7
 - Document Object Model See DOM
 - document type definition See DTD
 - documentation
 - comments on.....xvii
 - DOM.....70
 - downloading the NETCONF Java toolkit.....41
 - DTD
 - defined.....10
 - separate for each Junos OS module.....175
 - E**
 - edit-config tag (NETCONF).....129
 - usage guidelines.....83
 - edit-path tag (Junos XML protocol)
 - database-status tag.....150
 - end-session tag (Junos XML protocol).....151
 - entity references, predefined (Junos XML).....13
 - error messages
 - from NETCONF server.....70
 - specifying handling during configuration
 - changes.....93
 - error-info tag (NETCONF).....138
 - usage guidelines.....70
 - error-message tag (NETCONF).....140
 - usage guidelines.....70
 - error-option tag (NETCONF)
 - usage guidelines.....93
 - error-path tag (NETCONF).....140
 - usage guidelines.....70
 - error-severity tag (NETCONF).....140
 - usage guidelines.....70
 - examples, Junos XML
 - mapping of configuration statement to tag
 - comments in configuration.....21
 - hierarchy levels.....16
 - identifier.....17

leaf statement with keyword and value.....	18	get-checksum-information tag (Junos XML protocol).....	152
leaf statement with keyword only.....	18	get-config tag (NETCONF).....	133
multiple options on multiple lines.....	20	usage guidelines	
multiple options on single line.....	20	all objects of type.....	191
multiple predefined values for option.....	19	complete configuration.....	188
multiple user-defined values for option.....	19	hierarchy level.....	189
examples, NETCONF		identifiers only.....	194
committing with confirmation.....	116	multiple elements.....	203
comparing rollback configurations.....	207	overview.....	184
creating configuration elements.....	100	single object.....	197
deleting		specific children.....	199
fixed-form option.....	104	get-configuration tag (Junos XML protocol).....	152
single configuration object.....	103	get-rescue-information tag (Junos XML).....	208
value from list of multiple values.....	105	get-rollback-information tag (Junos XML)	
merging in new configuration data.....	98	comparing previous configurations.....	206
providing configuration data		displaying previous configuration tag.....	204
in a stream.....	87	get-xnm-information tag (Junos XML).....	210
replacing configuration elements.....	107	groups attribute (Junos XML protocol)	
requesting		get-configuration tag.....	152
all objects of a type.....	193		
one configuration level.....	190	H	
previous (rollback) configuration.....	205	hello tag (NETCONF).....	138
XML schema.....	211	usage guidelines.....	61
session.....	76		
terminating session.....	75	I	
exclusive tag (Junos XML protocol)		identifiers	
database-status tag.....	150	Junos XML mapping.....	17
		idle-time tag (Junos XML protocol)	
F		database-status tag.....	150
file-checksum attribute		inherit attribute (Junos XML protocol)	
checksum-information tag.....	144	get-configuration tag.....	152
files		input-file attribute	
junos.xsd.....	211	checksum-information tag.....	144
filter tag (NETCONF)		install-prereqs script	
requesting information.....	132, 133	Perl client applications (NETCONF).....	37
usage guidelines.....	188	installing the NETCONF Java toolkit.....	41
font conventions.....	xvi	interface-ranges attribute (Junos XML protocol)	
force-synchronize tag (Junos XML protocol).....	145	get-configuration tag.....	152
format attribute (Junos XML protocol)			
get-configuration tag.....	152	J	
load-configuration tag.....	156	JSON format	
format tag (Junos XML).....	204	displaying command output in.....	152, 175, 177
		Junos OS	
G		XML.....	7
get tag (NETCONF).....	132	Junos XML	
usage guidelines		in configuration statements.....	88
overview.....	183		

- Junos XML API
 - advantages of.....4
 - overview.....3
 - predefined entity references.....13
- Junos XML protocol server
 - parsing output from.....70
- Junos XML tags
 - compare tag.....206
 - configuration.....16
 - attributes used.....186
 - configuration-information tag
 - comparing configurations.....206
 - displaying configuration.....204
 - configuration-output tag
 - comparing configurations.....206
 - displaying configuration.....204
 - displaying CLI output as.....7
 - format tag.....204
 - get-rescue-information tag.....208
 - get-rollback-information tag
 - comparing previous configurations.....206
 - displaying previous configuration.....204
 - get-xnm-information.....210
 - junos:comment tag.....20
 - mapping
 - command options, fixed-form.....15
 - command options, variable.....15
 - configuration, comments.....20
 - configuration, hierarchy level.....16
 - configuration, identifier.....17
 - configuration, multiple multi-option
 - lines.....19
 - configuration, multivalue leaf.....19
 - configuration, single-value leaf.....18
 - namespace.....210
 - notational conventions.....9
 - output tag.....177
 - rollback tag
 - comparing configurations.....206
 - displaying configuration.....204
 - rollback-information tag
 - comparing configurations.....206
 - displaying configuration.....204
 - type.....210
 - undocumented.....184
 - xsd:import.....211
 - xsd:schema.....210
 - junos.xsd file.....211
 - junos:changed-localtime attribute (Junos XML).....167
 - usage guidelines.....186
 - junos:changed-seconds attribute (Junos XML).....167
 - usage guidelines.....186
 - junos:comment tag (Junos XML).....20
 - junos:commit-localtime attribute (Junos XML).....168
 - usage guidelines.....186
 - junos:commit-seconds attribute (Junos XML).....168
 - usage guidelines.....186
 - junos:commit-user attribute (Junos XML).....169
 - usage guidelines.....186
- K**
 - keyword in configuration statement, Junos XML
 - mapping17
 - kill-session tag (NETCONF).....134
 - usage guidelines.....74
- L**
 - leaf statement
 - Junos XML mapping.....18
 - load-configuration tag (Junos XML protocol).....156
 - load-configuration-results tag (Junos XML protocol).....159
 - load-error-count tag (Junos XML protocol).....159
 - load-success tag (Junos XML protocol).....149, 159
 - lock tag (NETCONF).....72, 134
 - lock-configuration tag (Junos XML protocol).....160
 - log tag (Junos XML protocol).....145
- M**
 - manuals
 - comments on.....xvii
 - merge data mode
 - configuration changes.....91
- N**
 - name tag (Junos XML protocol).....162
 - namespace tag (Junos XML).....210
 - namespaces See XML, namespaces
 - NET::Netconf module
 - about
 - NET::Netconf module.....27
 - Net::Netconf module
 - downloading
 - Net::Netconf module.....35

NETCONF		O	
trace log files.....	45, 46	ok tag (NETCONF).....	139
NETCONF client applications		usage guidelines.....	69
overview.....	27	open-configuration tag (Junos XML protocol).....	161
NETCONF Java toolkit		operation attribute (Junos XML).....	169
downloading.....	41	usage guidelines	
installing.....	41	creating element.....	99
NETCONF over SSH.....	55	deleting element.....	101
Netconf Perl client		replacing element.....	106
downloading.....	35	operational mode, CLI	
NETCONF server		Junos XML mapping	
classes of responses emitted.....	67	for requests.....	14
closing connection to.....	75	for responses.....	67
connecting to.....	60	operational request	
error message from.....	70	format.....	177
establishing session with.....	61	options in configuration statements, Junos XML	
parsing output from.....	70	mapping.....	19
sending request to.....	65	outbound SSH.....	51
verifying compatibility with application.....	63	configuring device.....	56
warning from.....	70	enabling SSH on device.....	59
NETCONF session		initialization sequence.....	59
authentication and security.....	60	installing ssh client.....	58
ending.....	75	NETCONF access protocol.....	51
establishing.....	61	prerequisites.....	56
example.....	76	<i>See also</i> SSH service	
terminating.....	74	output from Junos XML protocol server,	
NETCONF session requirements.....	41	parsing.....	70
netconf statement.....	119	output from NETCONF server, parsing.....	70
NETCONF tags		output tag (Junos XML).....	177
notational conventions.....	9	overview	
NETCONF XML management protocol		XML.....	9
advantages of.....	4	P	
comments, treatment of.....	12	parentheses, in syntax descriptions.....	xvii
conventions.....	10	passwords, text-based	
overview.....	3	SSH service.....	53
server <i>See</i> NETCONF XML protocol server		path attribute (Junos XML protocol)	
tags.....	9	get-checksum-information tag.....	152
white space, treatment of.....	12	Perl client applications	
NETCONF XML management protocol session		closing connection to server.....	233
client role.....	24	collecting parameters interactively.....	219
overview.....	23	connecting to the NETCONF server.....	217
NETCONF XML protocol server.....	3	declaring contants.....	216
newline character in XML tag sequences.....	12	example.....	226, 227
no-change data mode		formatting responses.....	230
configuration changes.....	92	importing Perl modules.....	216
no-change mode (NETCONF).....	101	parsing responses.....	230
		submitting requests.....	222
		writing.....	215

- Perl client applications (NETCONF)
 - install-prereqs script.....37
 - installing.....36, 39
 - installing prerequisites.....37
 - prerequisite modules.....37
- pid tag (Junos XML protocol)
 - database-status tag.....150
- PKI key pair
 - SSH service.....53
- port statement
 - NETCONF-over-SSH.....120
- predefined entity references (Junos XML).....13
- prerequisites
 - NETCONF XML management protocol.....51
- process-disabled tag (Junos XML protocol).....161
- process-not-configured tag (Junos XML protocol).....161
- process-not-running tag (Junos XML protocol).....161
- R**
 - rate-limit statement.....121
 - reason tag (Junos XML protocol).....161
 - replace (NETCONF 'operation' attribute)
 - usage guidelines.....106
 - replace data mode
 - configuration changes.....91
 - replace-pattern attribute (Junos XML protocol).....170
 - usage guidelines.....108
 - request tags (XML) See tags (XML)
 - request-end-session tag (Junos XML protocol).....162
 - rescue attribute (Junos XML protocol)
 - load-configuration tag.....156
 - rescue configuration
 - displaying (NETCONF).....208
 - response tags (XML) See reusing for requests See tags (XML)
 - rfc-compliant.....79
 - rfc-compliant statement.....122
 - rollback attribute (Junos XML protocol)
 - load-configuration tag.....156
 - rollback tag (Junos XML)
 - comparing configurations.....206
 - displaying configuration.....204
 - rollback-information tag (Junos XML)
 - comparing configurations.....206
 - displaying configuration.....204
 - routing-engine tag (Junos XML protocol).....149, 162
- RPC
 - displaying command output in.....7, 14
- rpc tag (NETCONF).....140
 - usage guidelines.....65
- rpc-error tag (NETCONF).....134, 140
 - usage guidelines.....70
- rpc-reply tag (NETCONF).....141
 - NETCONF namespace.....79
 - usage guidelines.....67
- running tag (NETCONF)
 - requesting information.....133
 - usage guidelines.....186
- S**
 - SAX.....70
 - schema See XML schema
 - schemas
 - YANG format.....238
 - security
 - NETCONF session.....60
 - server See NETCONF XML protocol server
 - session See NETCONF XML management protocol
 - session See NETCONF XML management protocol session
 - session, NETCONF
 - requirements.....41
 - session-id tag (NETCONF)
 - initializing session.....138
 - usage guidelines.....61
 - terminating session.....134
 - usage guidelines.....74
 - show system schema command.....238
 - Simple API for XML See SAX
 - software versions
 - compatibility between NETCONF client and server.....63
 - source tag (NETCONF)
 - replacing entire configuration.....128
 - usage guidelines.....94
 - requesting configuration information
 - usage guidelines.....186
 - requesting information.....133
 - validating configuration.....135
 - usage guidelines.....113
 - space character in XML tag sequences.....12
 - SSH connection.....41
 - SSH service
 - client software.....52
 - connecting to device.....60

enabling on device.....	55
logging in.....	52
NETCONF access protocol.....	51
passwords, text-based.....	53
PKI key pair.....	53
prerequisites.....	52
ssh statement	
netconf.....	123
start-time tag (Junos XML protocol)	
database-status tag.....	150
streaming data	
configuration data.....	85
referencing configuration data.....	87
support, technical See technical support	
synchronize tag (Junos XML protocol).....	145
syntax conventions.....	xvi

T

tags See Junos XML tags, NETCONF tags	
tags (XML)	
Junos XML.....	9
NETCONF.....	9
request	
children of.....	11
defined.....	10
Junos XML.....	175, 177
NETCONF.....	183, 184
response	
children of.....	12
defined.....	10
Junos XML.....	175
NETCONF.....	67
reusing.....	21
rpc-reply as container for.....	183, 184
white space in and around.....	12
target tag (NETCONF).....	135, 141
unlocking configuration.....	72
usage guidelines	
replacing entire configuration.....	94
technical support	
contacting JTAC.....	xviii
terminal tag (Junos XML protocol)	
database-status tag.....	150
traceoptions statement	
NETCONF.....	45, 46
netconf.....	124
tracing operations	
NETCONF.....	45, 46
type tag (Junos XML).....	210

U

undocumented tag (Junos XML).....	184
unlock tag (NETCONF).....	73, 135
unlock-configuration tag (Junos XML	
protocol).....	163
url attribute (Junos XML protocol)	
load-configuration tag.....	156
url tag (NETCONF)	
changing configuration	
usage guidelines.....	85
replacing entire configuration.....	128
usage guidelines.....	94
user tag (Junos XML protocol)	
database-status tag.....	150

V

validate tag (NETCONF).....	135
usage guidelines.....	113

W

warning	
from NETCONF server.....	70
white space in XML tag sequences.....	12

X

XML	
, namespaces	
NETCONF.....	79
namespaces.....	175
defined for operational response tags.....	68
in NETCONF server replies.....	79
overview.....	9
schema, requesting.....	210
tags See Junos XML tags, NETCONF tags	
xmlns attribute.....	171
configuration tag	
usage guidelines.....	186
Junos XML operational responses	
usage guidelines.....	175
NETCONF	
usage guidelines.....	67
xnm:error tag (Junos XML protocol).....	164
xnm:warning tag (Junos XML protocol).....	165
xsd:import tag (Junos XML).....	211
xsd:schema tag (Junos XML).....	210

Y**YANG**

importing modules.....	235
modules.....	30
overview.....	29

YANG modules

configuration module.....	235
generating.....	238
importing.....	235
overview.....	30

