JUNIPER NETWORKS | Engineering Simplicity

Junos® OS

Junos XML Management Protocol Developer Guide

Published
2025-12-09

JUNOS

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

Junos® OS Junos XML Management Protocol Developer Guide

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 https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About This Guide

Use this guide to remotely operate and configure Junos devices using the Juniper Networks Junos XML Management Protocol.

1

PART

Overview

CHAPTER 1

Junos XML Management Protocol and Junos XML API Overview

IN THIS CHAPTER

Junos XML Management Protocol and Junos XML API Overview

The Junos XML Management Protocol is an Extensible Markup Language (XML)-based protocol that client applications use to manage the configuration on Junos devices. It uses an XML-based data encoding for the configuration data and remote procedure calls (RPCs). The Junos XML protocol defines basic operations that are equivalent to configuration mode commands in the CLI. Applications use the protocol operations to display, edit, and commit configuration statements (among other operations), just as administrators use CLI configuration mode commands such as `show`, `set`, and `commit` to perform those operations.

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

Client applications request information and change the configuration on a switch, router, or security device by encoding the request with tag elements from the Junos XML management protocol and Junos XML API and sending it to the Junos XML protocol server on the device. The Junos XML protocol server is integrated into the Junos operating system and does not appear as a separate entry in process listings. The Junos XML protocol server directs the request to the appropriate software modules within the device, encodes the response in Junos XML protocol 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 tag. The Junos XML protocol server gathers the

information from the interface process and returns it in the Junos XML API `<interface-information>` response tag element.

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

RELATED DOCUMENTATION

- *Advantages of Using the Junos XML Management Protocol and Junos XML API*
- *XML and Junos OS Overview*
- *XML Overview*

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

IN THIS SECTION

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

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

Parsing Device Output

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

The formatted ASCII follows:

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

The corresponding XML-tagged version is:

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

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

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

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

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

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

Displaying Device Output

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

RELATED DOCUMENTATION

Junos XML Management Protocol and Junos XML API Overview

XML Overview

CHAPTER 2

Junos XML Protocol and Junos XML Tags Overview

IN THIS CHAPTER

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 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.

To display the configuration or operational mode command output as 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. The tag

elements that describe Junos OS configuration or operational data belong to the Junos XML API, which defines the Junos OS 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 request tag 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 Junos XML API request tag 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 the Junos XML API `<get-alarm-information>` request tag and sends the XML request to the Junos OS 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 OS 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 information pertaining to the Junos OS environment, and some scripts receive additional script-specific input that depends on the script type.

RELATED DOCUMENTATION

- Junos XML API Explorer

XML Overview

IN THIS SECTION

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

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

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

The following sections discuss general aspects of XML.

Tag Elements

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

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

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

As the preceding examples show, angle brackets enclose the name of the element. This is an XML convention, and the brackets are a required part of the complete element name. They are not to be confused with the angle brackets used in the Juniper Networks documentation to indicate optional parts of Junos OS CLI command strings.

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

When discussing XML elements in text, this documentation conventionally uses just the opening tag to represent the complete 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.

Attributes

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

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

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

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

Namespaces

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

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

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

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

During processing, the prefix is expanded into the URI reference. Although there may be multiple namespaces that define an `output` element or function, the use of `jcs:output` explicitly defines which `output`

function is used. You can choose any prefix to refer to the contents in a namespace, but there must be an existing declaration in the XML document that binds the prefix to the associated URI.

Document Type Definition

An XML-tagged document or data set is *structured* because a set of rules specifies the ordering and interrelationships of the items in it. A file called a *document type definition*, or *DTD*, defines these rules. The rules define the contexts in which each tagged item can—and in some cases must—occur. A DTD:

- Lists every element that can appear in the document or data set
- Defines the parent-child relationships between the tags
- Specifies other tag characteristics

The same DTD can apply to many XML documents or data sets.

RELATED DOCUMENTATION

Junos XML Management Protocol and Junos XML API Overview

XML and Junos OS Overview

XML and Junos XML Management Protocol Conventions Overview

IN THIS SECTION

A client application must comply with XML and Junos 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 Junos XML protocol and Junos XML document type definitions (DTDs) for the kind of information encoded in the request. The client application must emit tag elements in the required order and only in legal contexts. Compliant applications are easier to maintain in the event of changes to the Junos OS or Junos XML protocol.

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

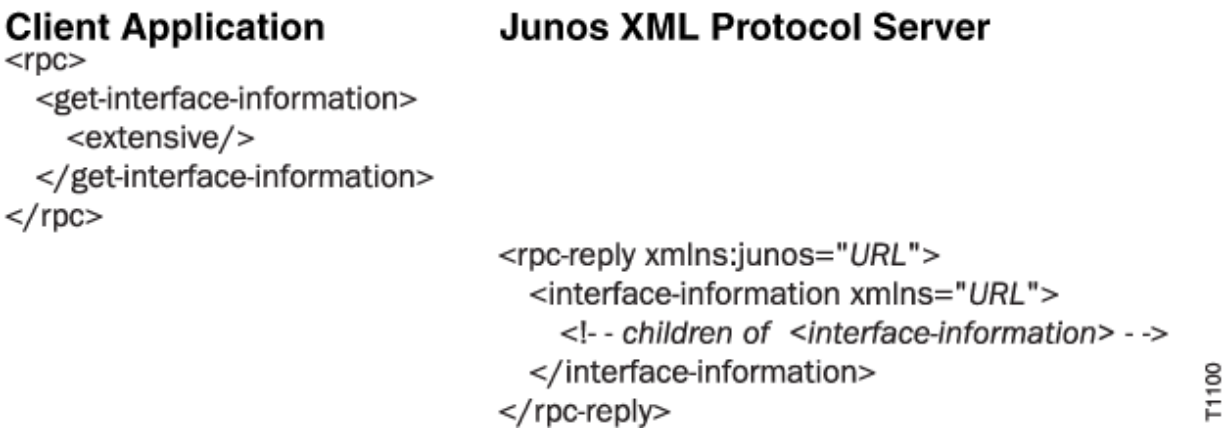
The following sections describe Junos XML management protocol conventions:

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 Requests to the Junos XML Protocol Server" on page 84.

A *response* tag element represents the Junos XML protocol 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 Junos XML Protocol Server Response" on page 87.

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



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 Junos XML protocol 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 14.

For information about the attributes in the opening `rpc-reply` tag, see "Parsing the Junos XML Protocol Server Response" on page 87. For information about the `xmlns` attribute in the opening `<interface-information>` tag, see "Requesting Operational Information Using the Junos XML Protocol" on page 359.

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>`.

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 Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol server preserves the white space in the configuration database.

In its responses, the Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol server gracefully but must not depend on their content. Client applications also cannot use comments to convey information to the Junos XML protocol 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. -->
```

XML Processing Instructions

An XML processing instruction (PI) contains information relevant to a particular protocol and has the following form:

```
<?PI-name attributes?>
```

Some PIs emitted during a Junos XML protocol session include information that a client application needs for correct operation. A prominent example is the `<?xml?>` element, which the client application and Junos XML protocol server each emit at the beginning of every Junos XML protocol session to specify which version of XML and which character encoding scheme they are using. For more information, see "Starting Junos XML Protocol Sessions" on page 75.

The Junos XML protocol server can also emit PIs that the client application does not need to interpret (for example, PIs intended for the CLI). If the client application does not understand a PI, it must treat the PI like a comment instead of exiting or generating an error message.

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 Junos XML protocol 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 1 on page 15 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 1: Predefined Entity Reference Substitutions for Tag Content Values

Disallowed Character	Predefined Entity Reference
& (ampersand)	`&amp;`

Table 1: Predefined Entity Reference Substitutions for Tag Content Values *(Continued)*

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

Table 2 on page 16 summarizes the mapping between disallowed characters and predefined entity references for attribute values.

Table 2: Predefined Entity Reference Substitutions for Attribute Values

Disallowed Character	Predefined Entity Reference
& (ampersand)	`&`
' (apostrophe)	`&apos;`
>> (greater-than sign)	`>`
< (less-than sign)	`<`
" (quotation mark)	`&quot;`

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

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

The `<condition>` 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> 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">
```

Map Junos OS Commands and Command Output to Junos XML Tag Elements

IN THIS SECTION

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 NETCONF and Junos XML protocol sessions to request information from a device running Junos OS or a device running Junos OS Evolved. The server returns the response using Junos XML tag elements enclosed within the response tag element. For example, the show interfaces command maps to the <get-interface-information> request tag, and the server returns the <interface-information> response tag.

The following sections outline how to map commands, command options, and command output to Junos XML tag elements.

Mapping Command Output to Junos XML Elements

On the Junos OS CLI, to display command output as Junos XML elements instead of as the default formatted ASCII text, include the | display xml option after the command. The XML elements that describe the Junos OS configuration or operational data belong to the Junos XML API. The Junos XML

API defines the Junos OS content that can be retrieved and manipulated by NETCONF and Junos XML management protocol operations.

The following example shows the output from the `show chassis hardware` command. The output is identical to the server's response for the `<get-chassis-inventory>` RPC request.

```
user@host> show chassis hardware | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/25.2R1.9/junos">
    <chassis-inventory xmlns="http://xml.juniper.net/junos/25.2R0/junos-chassis">
        <chassis junos:style="inventory">
            <name>Chassis</name>
            <serial-number>JN1085AA1AFA</serial-number>
            <description>MX960</description>
            <chassis-module>
                <name>Midplane</name>
                <version>REV 02</version>
                <part-number>710-013698</part-number>
                <serial-number>AA0001</serial-number>
                <description>MX960 Backplane</description>
                <model-number>CHAS-BP-MX960-S</model-number>
            </chassis-module>
            <chassis-module>
                <name>FPM Board</name>
                <version>REV 02</version>
                <part-number>710-014974</part-number>
                <serial-number>AA0002</serial-number>
                <description>Front Panel Display</description>
                <model-number>CRAFT-MX960-S</model-number>
            </chassis-module>
            <chassis-module>
                <name>PDM</name>
                <version>Rev 02</version>
                <part-number>740-013110</part-number>
                <serial-number>AAA0000001A</serial-number>
                <description>Power Distribution Module</description>
            </chassis-module>
            <!-- other child tags of <chassis> -->
        </chassis>
    </chassis-inventory>
    <cli>
        <banner></banner>
```

```
    </cli>
</rpc-reply>
```

Mapping Commands to Junos XML Request Tag Elements

You can find information about the available Junos OS or Junos OS Evolved operational mode commands and their equivalent Junos XML RPC request tags using the following methods:

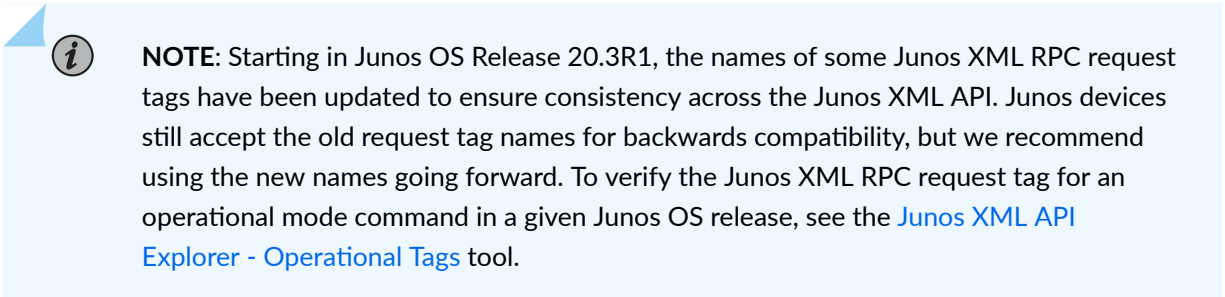
- Appending `| display xml rpc` to an operational command in the CLI.
- Using the Junos XML API Explorer - Operational Tags application to search for a command or request tag in a given release.

You can use the Junos XML API Explorer tool to: verify a command, map the command to its equivalent Junos XML RPC request tag and child tags, and view the expected response tag for various Junos OS or Junos OS Evolved releases.

In the Junos OS CLI, you can display the Junos XML request tag elements for any operational mode command that has a Junos XML counterpart. To display the request tags for a given command, enter the command and pipe it to the `display xml rpc` command.

The following example displays the RPC tags for the `show route` command:

```
user@host> show route | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/25.2R1.9/junos">
    <rpc>
        <get-route-information>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```



NOTE: Starting in Junos OS Release 20.3R1, the names of some Junos XML RPC request tags have been updated to ensure consistency across the Junos XML API. Junos devices still accept the old request tag names for backwards compatibility, but we recommend using the new names going forward. To verify the Junos XML RPC request tag for an operational mode command in a given Junos OS release, see the Junos XML API Explorer - Operational Tags tool.

Mapping for Command Options with Variable Values

Many CLI commands have options that identify the object that the command affects or reports on, 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 find 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. Alternatively, issue the command and command option in the CLI and append the `| display xml rpc` option.

Table 3 on page 20 shows the Junos XML tag elements for two operational commands that have variable-form options. In the `show interfaces` command, ge-0/0/1 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.

Table 3: Commands with Variable-Form Options

Command	Junos XML Tags

`show interfaces ge-0/0/1`	`<rpc>`
	`  <get-interface-information>`
	`  <interface-name>ge-0/0/1</interface-name>`
	`  </get-interface-information>`
	`</rpc>`

`show bgp neighbor 10.168.1.122`	`<rpc>`
	`  <get-bgp-neighbor-information>`
	`  <neighbor-address>10.168.1.122</neighbor-address>`
	`  </get-bgp-neighbor-information>`
	`</rpc>`

You can display the Junos XML RPC tags for a command and its options in the CLI by executing the command and command option and appending `| display xml rpc`.

```
user@host> show interfaces ge-0/0/1 | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/25.2R1.9/junos">
    <rpc>
        <get-interface-information>
            <interface-name>ge-0/0/1</interface-name>
        </get-interface-information>
    </rpc>
    <cli>
```

```
        <banner></banner>
    </cli>
</rpc-reply>
```

Mapping for Fixed-Form Command Options

Some CLI commands include options that have a fixed form, such as the `brief` and `detail` options, 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`:

To view the tags in the CLI:

```
user@host> show isis adjacency detail | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/25.2R1.9/junos">
    <rpc>
        <get-isis-adjacency-information>
            <detail/>
        </get-isis-adjacency-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
20.3R1	Starting in Junos OS Release 20.3R1, the names of some Junos XML RPC request tags have been updated to ensure consistency across the Junos XML API.

Map Configuration Statements to Junos XML Tag Elements

IN THIS SECTION

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.

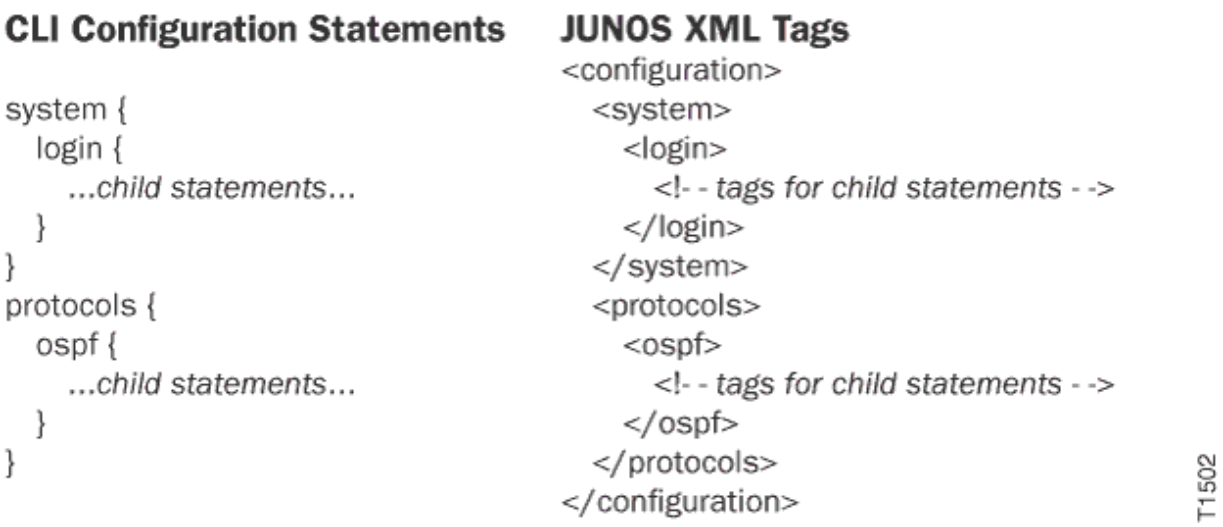
NOTE: For some configuration statements, the notation used when you type the statement at the CLI configuration-mode prompt differs from the notation used in a configuration file. The same Junos XML tag element maps to both notational styles.

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

Mapping for Hierarchy Levels and Container Statements

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.



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 25, "Mapping for Leaf Statements with Multiple Values" on page 26, and "Mapping for Multiple Options on One or More Lines" on page 27.

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

```
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;


        }
    }
}
```

JUNOS XML Tags

```
<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>
```

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

```
system {
   login {
      message "Authorized users only";
      ...other statements under login...
   }
}
protocols {
   ospf {
      preference 15;
      ...cther statements under ospf...
   }
}
```

JUNOS XML Tags

```
<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>
```

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

```
forwarding-options {
   sampling {
      disable;
      ...other statements under sampling ...
   }
}
```

JUNOS XML Tags

```
<configuration>
   <forwarding-options>
      <sampling>
         <disable/>
         <!- - tags for other child statements - ->
      </sampling>
   </forwarding-options>
</configuration>
```

T1505

Mapping for Leaf Statements with Multiple Values

Some Junos OS 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

```
protocols {
   bgp {
      group 23 {

         import [ policy1 policy2 ];

      }
   }
}
```

JUNOS XML Tags

```
<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

```
system {
   login {
      class user-accounts {

         permissions [ configure admin control ];


      }
   }
}
```

JUNOS XML Tags

```
<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 OS 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 10.0.01 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 OS configuration can include comments that describe statements in the configuration. In CLI configuration mode, the `annotate` command defines 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

```
protocols {
    ospf {

        /* New backbone area */
        area 0.0.0.0 {

            interface so-0/0/0 { # From jnpr1 to jnpr2

                hello-interval 5;
            }
        }
    }
}
```

JUNOS XML Tags

```
<configuration>
  <protocols>
    <ospf>
      <junos:comment>
          /* New backbone area */
      </junos:comment>
      <area>
        <name>0.0.0.0</name>
        <junos:comment> # From jnpr1 to jnpr2</junos:comment>
        <interface>
          <name>so-0/0/0</name>
          <hello-interval>5</hello-interval>
        </interface>
      </area>
    </ospf>
  </protocols>
</configuration>
```

T1510

Using Configuration Response Tag Elements in Junos XML Protocol Requests and Configuration Changes

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

The Junos XML tag elements within the <configuration> 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 Junos XML protocol 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 Junos XML protocol server's response to 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 Junos XML protocol server's response, it is certain to be syntactically correct. For more information about changing routing platform configuration, see "Requesting Configuration Changes Using the Junos XML Protocol" on page 206.

Similarly, when a client application requests information about a configuration element (hierarchy level or configuration object), it uses the same elements that the Junos XML protocol server will return in response. To represent the element, the client application sends a complete stream of elements from the top of the configuration hierarchy (represented by the <configuration> tag) down to the requested element. The innermost element, which represents the level or object, is either empty or includes the identifier tag only. The Junos XML protocol 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 the Junos XML Protocol" on page 375.

The tag streams emitted by the Junos XML protocol server and by a client application can differ in the use of white space, as described in "XML and Junos XML Management Protocol Conventions Overview" on page 11.

RELATED DOCUMENTATION

XML and Junos XML Management Protocol Conventions Overview | 11

Map Configuration Statements to Junos XML Tag Elements | 22

CHAPTER 3

Junos XML Protocol and JSON Overview

IN THIS CHAPTER

Map Junos OS Command Output to JSON in the CLI

Junos OS and Junos OS Evolved natively support XML for the operation and configuration of Junos devices. The CLI and the infrastructure communicate using XML. When you issue an operational command or display the configuration in the CLI, the CLI converts the output from XML into a readable text format for display.

Junos devices also support a JavaScript Object Notation (JSON) representation of the operational command output and the configuration hierarchy. To display the command output or configuration in JSON instead of in the default formatted ASCII text, append the `| display json` option to the command in the CLI.

The following example executes the `show chassis hardware` command and displays the output in JSON format. The response is identical to the NETCONF or Junos XML protocol server response for the `<get-chassis-inventory format="json">` RPC request.

```
user@host> show chassis hardware | display json
{
    "chassis-inventory" : [
    {
        "attributes" : {"xmlns" : "http://xml.juniper.net/junos/24.2R1/junos-chassis"},
        "chassis" : [
        {
            "attributes" : {"junos:style" : "inventory"},
            "name" : [
            {
                "data" : "Chassis"
```

```
            }
            ],
            "serial-number" : [
            {
                "data" : "XX00X00XXXXX"
            }
            ],
            "description" : [
            {
                "data" : "MX960"
            }
            ],
            "chassis-module" : [
            {
                "name" : [
                {
                    "data" : "Midplane"
                }
                ],
                "version" : [
                {
                    "data" : "REV 03"
                }
                ],
                "part-number" : [
                {
                    "data" : "710-013698"
                }
                ],
                "serial-number" : [
                {
                    "data" : "XX0001"
                }
                ],
                "description" : [
                {
                    "data" : "MX960 Backplane"
                }
                ],
                "model-number" : [
                {
                    "data" : "CHAS-BP-MX960-S"
                }
```

```
                ]
            },
            //* additional JSON objects *//
            ]
        }
        ]
    }
    ]
}
```

You can configure how a Junos device emits configuration data in JSON format. To configure the default export format, include the appropriate statement at the `[edit system export-format json]` hierarchy level. You can configure the JSON export format as one of the following:

- `ietf`—Emit JSON configuration data as defined in IETF RFC 7951, *JSON Encoding of Data Modeled with YANG*.
- `verbose`—Emit all configuration objects as JSON arrays.

By default, Junos devices emit JSON-formatted state data in non-compact format, which emits all objects as JSON arrays. In Junos OS Release 24.2 and earlier and Junos OS Evolved Release 24.2 and earlier, Junos devices support emitting the device's state data in compact JSON format, in which only objects that have multiple values are emitted as JSON arrays. To configure the device to emit compact JSON format in supported releases, configure the `compact` statement at the `[edit system export-format state-data json]` hierarchy level.

```
[edit]
user@host# set system export-format state-data json compact
```

The following example executes the `show system uptime` command and displays the output in both non-compact and compact JSON format.

```
user@host> show system uptime | display json
```

The following output shows the non-compact JSON format:

```
{
    "system-uptime-information" : [
    {
        "attributes" : {"xmlns" : "http://xml.juniper.net/junos/18.1R1/junos"},
        "current-time" : [
```

```
        {
            "date-time" : [
            {
                "data" : "2018-05-15 13:43:46 PDT",
                "attributes" : {"junos:seconds" : "1526417026"}
            }
            ]
        }
        ],
        "time-source" : [
        {
            "data" : " NTP CLOCK "
        }
        ],
        "system-booted-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-15 10:57:02 PDT",
                "attributes" : {"junos:seconds" : "1526407022"}
            }
            ],
            "time-length" : [
            {
                "data" : "02:46:44",
                "attributes" : {"junos:seconds" : "10004"}
            }
            ]
        }
        ],
        "protocols-started-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-15 10:59:33 PDT",
                "attributes" : {"junos:seconds" : "1526407173"}
            }
            ],
            "time-length" : [
            {
                "data" : "02:44:13",
                "attributes" : {"junos:seconds" : "9853"}
            }
```

```
            ]
        }
        ],
        "last-configured-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-02 17:57:44 PDT",
                "attributes" : {"junos:seconds" : "1525309064"}
            }
            ],
            "time-length" : [
            {
                "data" : "1w5d 19:46",
                "attributes" : {"junos:seconds" : "1107962"}
            }
            ],
            "user" : [
            {
                "data" : "admin"
            }
            ]
        }
        ],
        "uptime-information" : [
        {
            "date-time" : [
            {
                "data" : "1:43PM",
                "attributes" : {"junos:seconds" : "1526417026"}
            }
            ],
            "up-time" : [
            {
                "data" : "2:47",
                "attributes" : {"junos:seconds" : "10034"}
            }
            ],
            "active-user-count" : [
            {
                "data" : "1",
                "attributes" : {"junos:format" : "1 user"}
            }
```

```
            ],
            "load-average-1" : [
            {
                "data" : "0.49"
            }
            ],
            "load-average-5" : [
            {
                "data" : "0.19"
            }
            ],
            "load-average-15" : [
            {
                "data" : "0.10"
            }
            ]
        }
        ]
    }
    ]
}
```

The output for the same command in compact JSON format is:

```
{
    "system-uptime-information" :
    {
        "current-time" :
        {
            "date-time" : "2018-05-15 13:49:56 PDT"
        },
        "time-source" : " NTP CLOCK ",
        "system-booted-time" :
        {
            "date-time" : "2018-05-15 10:57:02 PDT",
            "time-length" : "02:52:54"
        },
        "protocols-started-time" :
        {
            "date-time" : "2018-05-15 10:59:33 PDT",
            "time-length" : "02:50:23"
        },
```

```
        "last-configured-time" :
        {
            "date-time" : "2018-05-15 13:49:40 PDT",
            "time-length" : "00:00:16",
            "user" : "admin"
        },
        "uptime-information" :
        {
            "date-time" : "1:49PM",
            "up-time" : "2:53",
            "active-user-count" : "1",
            "load-average-1" : "0.00",
            "load-average-5" : "0.06",
            "load-average-15" : "0.06"
        }
    }
}
```


Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
24.4R1 & 24.4R1-EVO	Starting in Junos OS Release 24.4R1 and Junos OS Evolved Release 24.4R1, we've deprecated the `compact` statement at the `[edit system export-format state-data json]` hierarchy level.

RELATED DOCUMENTATION

Map Junos OS Commands and Command Output to Junos XML Tag Elements | 17

Map Junos OS Configuration Statements to JSON

IN THIS SECTION

A configuration for a device running Junos OS is stored as a hierarchy of statements. The configuration statement hierarchy has two types of statements:

- *container statements*—statements that contain other statements
- *leaf statements*—statements that do not contain other statements

All of the container and leaf statements together form the *configuration hierarchy*.

The configuration hierarchy can be represented using JavaScript Object Notation (JSON) in addition to formatted ASCII text, Junos XML elements, and configuration mode `set` commands. You can view the configuration of a device running Junos OS in JSON format by executing the `show configuration | display json` command in the CLI. You can also load JSON-formatted configuration data on the device.

The following sections describe the mapping between the formatted ASCII text and the default format used for Junos OS configuration statements in JSON:

Mapping for Hierarchy Levels and Container Statements

The Junos OS configuration hierarchy is represented in JSON by a JSON object with a single top-level member, or name/value pair, that has the field name set to `"configuration"` and a value that contains a JSON object representing the entire configuration. The `configuration` member corresponds to the `[edit]` hierarchy level in CLI configuration mode. Most statements at the next few levels of the configuration hierarchy are container statements.

In JSON, each Junos OS hierarchy level or container statement is a member of its parent object. The member, or name/value pair, has a field name corresponding to the name of the hierarchy or container. Its value is a JSON object that contains members representing the child containers and leaf statements at that hierarchy level. The object might also contain a member that holds the list of attributes, if any, associated with that hierarchy.

The following example shows the mapping between formatted ASCII text and JSON for two statements at the top level of the configuration hierarchy:

CLI Configuration Statements

```
system {
   login {
      ...child statements...
   }
}
protocols {
   ospf {
      ...child statements...
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "system" : {
         "login" : {
            ...JSON configuration data...
         }
      }
      "protocols" : {
         "ospf" : {
            ...JSON configuration data...
         }
      }
   }
}
```

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 might consist of a keyword and identifier or just an identifier.

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

keyword is a fixed string that indicates the type of object being defined, and *identifier* is a unique name for an instance of that type. In the following example, `user` is a keyword, and *username* is the identifier.

```
user username {
    /* child statements */
}
```

In JSON, all instances of a configuration object are contained within a single name/value pair in which the field name generally matches the *keyword* string, and the value is an array of JSON objects, each of which is an instance of the configuration object. The JSON syntax differs from the CLI in its treatment of the identifier. In JSON, each instance of the configuration object uses a name/value pair for the identifier, where the field name distinguishes this data as the identifier, and the value is the actual unique identifier for the object. Most frequently, the field name is just `name`. Some objects have multiple identifiers, and might use a field name other than `name`. JSON data that specifies an identifier is always listed first within the corresponding object, but after any attribute list included for that object.

```
{
   "keyword" : [
   {
      "@" : {
         "comment" : "comment"
      },
      "name" : "identifier",
      JSON data for individual characteristics
   },
   {
      "name" : "identifier",
      JSON data for individual characteristics
   }
   ]
}
```

NOTE: 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-`.

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 JSON mapping for leaf statements, see "Mapping for Single-Value and Fixed-Form Leaf Statements" on page 41.

The following example shows the mapping of formatted ASCII text to JSON for configuration statements that define two BGP groups named G1 and G2. In the JSON syntax, the `group` member's value is an array that contains a separate JSON object for each BGP group.

CLI Configuration Statements

```
protocols {
   bgp {
      group G1 {
         type external;
         peer-as 64501;
         neighbor 10.0.0.1;
      }
      group G2 {
         type external;
         peer-as 64502;
         neighbor 10.0.10.1;
      }
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "protocols" : {
         "bgp" : {
            "group" : [
            {
               "name" : "G1",
               "type" : "external",
               "peer-as" : "64501",
               "neighbor" : [
               {
                  "name" : "10.0.0.1"
               }
               ]
            },
            {
```

```
                "name" : "G2",
                "type" : "external",
                "peer-as" : "64502",
                "neighbor" : [
                {
                   "name" : "10.0.10.1"
                }
                ]
             }
             ]
          }
       }
    }
}
```

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;
```

Junos OS leaf statements are mapped to name/value pairs in JSON. In general, the field name is the same as the `keyword` string, and the value is the same as the *`value`* string.

The following example shows the JSON mapping 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

```
system {
   login {
      message "Authorized users only.";
      ... other statements under login ...
   }
}
protocols {
   ospf {
      preference 15;
      ... other statements under ospf ...
```

```
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "system" : {
         "login" : {
            "message" : "Authorized users only.",
             ... JSON data for other statements under login ...
         }
      },
      "protocols" : {
         "ospf" : {
            "preference" : "15",
            ... JSON data for other statements under ospf ...
         }
      }
   }
}
```

Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. Junos OS represents such statements in JSON by setting the value in the name/value pair to `[null]`. The following example shows the JSON mapping for the `disable` statement at the `[edit forwarding-options sampling]` hierarchy level.

CLI Configuration Statements

```
forwarding-options {
   sampling {
      disable;
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "forwarding-options" : {
         "sampling" : {
```

```
            "disable" : [null]
         }
      }
   }
}
```

Mapping for Leaf Statements with Multiple Values

Some Junos OS 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 example:

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

As discussed in "Mapping for Single-Value and Fixed-Form Leaf Statements" on page 41, leaf statements are mapped to name/value pairs in JSON, where the field name is the same as the *keyword* string. To represent multiple values, the value portion of the JSON data uses an array of comma-separated strings.

The following example shows the JSON mapping for a CLI statement with multiple user-defined values. The `import` statement imports two routing policies defined elsewhere in the configuration.The formatted ASCII text uses a space-separated list of values, whereas the JSON data uses an array with a comma-separated list of strings.

CLI Configuration Statements

```
protocols {
   bgp {
      group 23 {
         import [ policy1 policy2 ];
      }
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "protocols" : {
         "bgp" : {
            "group" : [
```

```
            {
               "name" : "23",
               "import" : ["policy1", "policy2"]
            }
            ]
         }
      }
   }
}
```

The following example shows the JSON mapping 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

```
system {
   login {
      class user-accounts {
         permissions [ admin configure control ];
      }
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "system" : {
         "login" : {
            "class" : [
            {
               "name" : "user-accounts",
               "permissions" : ["admin", "configure", "control"]
            }
            ]
         }
      }
   }
}
```

Mapping for Multiple Options on One or More Lines

For some Junos OS 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.

In JSON, the same configuration object maps to a name/value pair. The field name is the same as the object name, and the value is a JSON object containing the options, which are represented by name/value pairs. If the first option has no keyword in the CLI statement, the JSON mapping assigns a name, which is equivalent to the option name defined in the schema and used for the corresponding Junos XML tag name.

The following example shows the JSON mapping for a CLI configuration statement with multiple options on a single line. The JSON representation of the `[edit system backup-router]` statement uses name/value pairs for both options and assigns the field name `address` for the first option (10.0.0.1), which has no CLI keyword.

CLI Configuration Statements

```
system {
   backup-router 10.0.0.1 destination 10.0.0.2/32;
}
```

JSON Syntax

```
{
   "configuration" : {
      "system" : {
         "backup-router" : {
            "address" : "10.0.0.1",
            "destination" : ["10.0.0.2/32"]
         }
      }
   }
}
```

The syntax for some configuration objects includes more than one multi-option line. In JSON, the configuration object maps to a name/value pair, as in the previous case. The field name is the same as the object name, and the value is a JSON object containing the options, which are represented by name/value pairs. For each option, the field name is the same as the option name, and the value is a JSON data structure that appropriately represents the configuration data for that option. When an option uses the same keyword but spans multiple lines, the JSON representation combines the options into a single

name/value pair. In this case, the value is an array of JSON objects in which each option is mapped to a separate object.

The following example shows the JSON mapping for the `traceoptions` statement at the `[edit protocols isis]` hierarchy level. The `traceoptions` statement has three child statements, each with multiple options. The CLI notation displays the individual `flag` options on separate lines, but the JSON representation combines the `flag` details into a single name/value pair. The value is an array of objects where each object contains the details for one flag.

CLI Configuration Statements

```
protocols {
   isis {
      traceoptions {
         file trace-file size 3m files 10 world-readable;
         flag route detail;
         flag state receive;
      }
   }
}
```

JSON Syntax

```
{
   "configuration" : {
      "protocols" : {
         "isis" : {
            "traceoptions" : {
               "file" : {
                  "filename" : "isis-trace-file",
                  "size" : "3m",
                  "files" : 10,
                  "world-readable" : [null]
               },
               "flag" : [
               {
                  "name" : "route",
                  "detail" : [null]
               },
               {
                  "name" : "state",
                  "receive" : [null]
```

```
            }
            ]
         }
      }
   }
}
```

Mapping for Attributes

The Junos OS configuration hierarchy can contain tags that modify a hierarchy or statement. For example, if you issue the `deactivate` command to deactivate a statement in the configuration, the `inactive:` tag is prepended to the item in the configuration to indicate this property. The Junos XML API represents this property as an attribute in the opening tag of the XML element.

The JSON representation of the Junos OS configuration uses metadata annotations to represent these properties. The metadata annotations are encoded as members of a single JSON object and include the "@" symbol as or within the name.

The metadata object representing the attribute list for a container statement is added as a new member of that object. The metadata object is placed directly inside the container object it modifies and uses a single "@" symbol as the member name. The metadata object representing the attribute list for a leaf statement is added as a sibling name/value pair that is placed directly after the statement it modifies and that has a member name that is the concatenation of the "@" symbol and the statement name. The metadata object value in both cases is an object containing name/value pairs that describe each of the attributes associated with that container or leaf statement.

```
{
   "container" : {
      "@" : {
         "attribute-name" : attribute-value,
         "attribute-name" : attribute-value
      },
      "statement-name" : "statement-value",
      "@statement-name" : {
         "attribute-name" : attribute-value,
         "attribute-name" : attribute-value
      }
   }
}
```

In the following examples, both the `[edit commit]` hierarchy and the `persist-groups-inheritance` statement have been deactivated. In the CLI, the statements are preceded by the `inactive:` tag. The Junos XML representation includes the `inactive="inactive"` attribute in each of the opening tags for those elements. The JSON mapping stores the attributes in an attribute list. The attribute list for the `[edit commit]` hierarchy is indicated with "@", because it is a container statement. The attribute list for the `persist-groups-inheritance` statement is indicated using "@persist-groups-inheritance", because it is a leaf statement.

CLI Configuration Statements

```
system {
   inactive: commit {
      inactive: persist-groups-inheritance;
   }
}
```

XML Syntax

```
<configuration>
   <system>
      <commit inactive="inactive">
         <persist-groups-inheritance inactive="inactive"/>
      </commit>
   </system>
</configuration>
```

JSON Syntax

```
{
   "configuration" : {
      "system" : {
         "commit" : {
            "@" : {
               "inactive" : true
            },
            "persist-groups-inheritance" : [null],
            "@persist-groups-inheritance" : {
               "inactive" : true
            }
         }
      }
```

```
   }
}
```

The attribute list for a specific instance of an object is similar to the attribute list for a container in that it is a name/value pair where the field name is a single "@" symbol, and the value is an object containing name/value pairs that describes each of the attributes. The attribute list is enclosed within the JSON object that identifies that instance and is the first member in the object, followed by the identifier for the object.

```
{
   "keyword" : [
   {
      "@" : {
         "attribute-name" : attribute-value
      },
      "name" : "identifier",
      ...JSON data for individual characteristics...
   },
   /* additional objects */
   ]
}
```

In the following example, the ge-0/0/0 interface is protected. In the CLI, the object is preceded by the `protect:` tag. The Junos XML representation includes the `protect="protect"` attribute in the opening tag for that object. The JSON mapping stores the `"protect" : true` attribute in an attribute list that is included in the JSON object identifying that specific interface.

CLI Configuration Statements

```
protect: ge-0/0/0 {
    unit 0 {
        family inet {
            address 198.51.100.1/24;
        }
    }
}
```

XML Syntax

```
<configuration>
   <interfaces>
```

```
      <interface protect="protect">
         <name>ge-0/0/0</name>
         <unit>
            <name>0</name>
            <family>
               <inet>
                  <address>
                     <name>198.51.100.1/24</name>
                  </address>
               </inet>
            </family>
         </unit>
      </interface>
   </interfaces>
</configuration>
```

JSON Syntax

```
{
   "configuration" : {
      "interfaces" : {
         "interface" : [
         {
            "@" : {
               "protect" : true
            },
            "name" : "ge-0/0/0",
            "unit" : [
            {
               "name" : 0,
               "family" : {
                  "inet" : {
                     "address" : [
                     {
                        "name" : "198.51.100.1/24"
                     }
                     ]
                  }
               }
            }
            ]
         }
```

```
        ]
      }
   }
}
```

Mapping for Configuration Comments

A Junos OS configuration can include comments that describe statements in the configuration. Configuration data formatted using ASCII text or Junos XML elements displays comments on the line preceding the statement that the comment modifies. In Junos XML format, the comment string is enclosed in a `<junos:comment>` element.

Comments are indicated using one of two delimiters: the paired strings `/*` and `*/` enclosing the comment, or the `#` character preceding the comment. You can use either delimiter in the comment string when you insert comments in the configuration. If you omit the delimiter, Junos OS automatically inserts `/*` and `*/`.

NOTE: Junos OS preserves any white space included in the comment.

Junos OS configuration data formatted using JSON maps a comment to a name/value pair that is stored as an attribute of the statement that it modifies. The field name is set to `"comment"`, and the value is the comment text string. The comment text string can include either of the two delimiters that indicate a comment. If you omit the delimiter from the comment text string when you load the JSON configuration data, Junos OS automatically adds the `/*` and `*/` delimiters to the comment. You can also create multiline comments in JSON configuration data by inserting the newline character (`\n`) in the comment string.

The following example shows the formatted ASCII configuration and corresponding JSON syntax for three comments. The example associates one comment with a hierarchy, another comment with an object that has an identifier, and a third comment with a leaf statement.

CLI Configuration Statements

```
protocols {
    /* New backbone area */
    ospf {
        area 0.0.0.0 {
            /* From jnpr1
            to jnpr2 */
            interface so-0/0/0.0 {
                # set by admin
                hello-interval 5;
            }
```

```
        }
    }
}
```

JSON Syntax

```
{
   "configuration" : {
      "protocols" : {
         "ospf" : {
            "@" : {
               "comment" : "/* New backbone area */"
            },
            "area" : [
            {
               "name" : "0.0.0.0",
               "interface" : [
               {
                  "@" : {
                     "comment" : "/* From jnpr1 \n to jnpr2 */"
                  },
                  "name" : "so-0/0/0.0",
                  "hello-interval" : 5,
                  "@hello-interval" : {
                     "comment" : "# set by admin"
                  }
               }
               ]
            }
            ]
         }
      }
   }
}
```

2

PART

Manage Junos XML Protocol Sessions

CHAPTER 4

Junos XML Protocol Session Overview

IN THIS CHAPTER

Junos XML Protocol Session Overview

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

Each request from the client application and each response from the Junos XML protocol server must constitute a *well-formed* XML document by obeying the structural rules defined in the Junos XML protocol and Junos XML document type definition (DTD) for the kind of information they encode. The client application must produce a well-formed XML document for each request by emitting tag elements in the required order and only in the legal contexts.

Client applications access the Junos XML protocol server using one of the protocols listed in "Supported Access Protocols for Junos XML Protocol Sessions" on page 55. To authenticate with the Junos XML protocol server, a client application uses either a Junos XML protocol-specific mechanism or the access protocol's standard authentication mechanism, depending on the protocol. After authentication, the Junos XML protocol server uses the Junos OS login usernames and classes configured on the device to determine whether a client application is authorized to make each request.

The following list outlines the basic structure of a Junos XML protocol session:

1. The client application establishes a connection to the Junos XML protocol server and opens the Junos XML protocol session.
2. The Junos XML protocol server and client application exchange initialization information, which is used to determine if they are using compatible versions of the Junos OS and the Junos XML management protocol.

3. The client application sends one or more requests to the Junos XML protocol server and parses its responses.
4. The client application closes the Junos XML protocol session and the connection to the Junos XML protocol server.

RELATED DOCUMENTATION

Supported Access Protocols for Junos XML Protocol Sessions | 55
Satisfy the Prerequisites for Establishing a Connection to the Junos XML Protocol Server | 60
Understanding the Client Application's Role in a Junos XML Protocol Session | 56

Supported Access Protocols for Junos XML Protocol Sessions

To connect to the Junos XML protocol server, client applications can use the access protocols and associated authentication mechanisms listed in Table 4 on page 55.

Table 4: Supported Access Protocols and Authentication Mechanisms

Access Protocol	Authentication Mechanism
cleartext, a Junos XML protocol-specific access protocol for sending unencrypted text over a TCP connection	Junos XML protocol-specific
SSH	Standard SSH
Outbound SSH	Outbound SSH
Secure Sockets Layer (SSL)	Junos XML protocol-specific
Telnet	Standard Telnet

We recommend using SSH and SSL because these protocols encrypt security information (such as passwords) before transmitting it across the network. Outbound SSH allows you to create an encrypted connection to the device in situations where you cannot connect to the device using standard SSH. The cleartext and Telnet protocols do not encrypt information.

For information about the prerequisites for each access protocol, see "Satisfying the Prerequisites for Establishing a Connection to the Junos XML Protocol Server" on page 60. For authentication instructions, see "Authenticating with the Junos XML Protocol Server for Cleartext or SSL Connections" on page 80.

RELATED DOCUMENTATION

Understanding the Client Application's Role in a Junos XML Protocol Session

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

1. Satisfies any prerequisites required for a connection, as described in "Satisfying the Prerequisites for Establishing a Connection to the Junos XML Protocol Server" on page 60.

2. Establishes a connection to the Junos XML protocol server on the routing, switching, or security platform, as described in "Connecting to the Junos XML Protocol Server" on page 73.

3. Starts a Junos XML protocol session, as described in "Starting Junos XML Protocol Sessions" on page 75.

4. Optionally locks the candidate configuration, creates a private copy of the configuration, or opens an instance of the ephemeral configuration database.

   Locking the configuration prevents other users or applications from changing it at the same time. Creating a private copy of the configuration enables the application to make changes without affecting the candidate configuration until the copy is committed. For more information, see "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

   For information about the ephemeral configuration database, see "Understanding the Ephemeral Configuration Database" on page 310 and "Enabling and Configuring Instances of the Ephemeral Configuration Database" on page 328.

5. Requests operational or configuration information, or changes the configuration, as described in "Sending Requests to the Junos XML Protocol Server" on page 84.

6. (Optional) Verifies the syntactic correctness of the candidate configuration or private copy before attempting to commit it, as described in "Verifying Configuration Syntax Using the Junos XML Protocol" on page 287.

7. Commits changes made to the candidate configuration or private copy, as described in "Committing the Candidate Configuration Using the Junos XML Protocol" on page 288, or commits changes made to an open instance of the ephemeral configuration database, as described in "Committing and Synchronizing Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol" on page 339.

8. Unlocks the candidate configuration if it is locked or closes a private copy of the configuration or an open instance of the ephemeral configuration database.

   Other users and applications cannot change the candidate configuration while it remains locked. For more information, see "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

9. Ends the Junos XML protocol session and closes the connection to the device, as described in "Ending a Junos XML Protocol Session and Closing the Connection" on page 100.

RELATED DOCUMENTATION

Supported Access Protocols for Junos XML Protocol Sessions | 55

Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections | 80

Parse the Junos XML Protocol Server Response | 87

Sample Junos XML Protocol Session | 101

Understanding the Request Procedure in a Junos XML Protocol Session

You can use the Junos 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 for many commands in the CLI `show` family of commands.

The tag for configuration requests is the Junos XML protocol `<get-configuration>` tag. 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 Junos XML protocol 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 Junos XML protocol server, a client application performs the procedures described in the indicated sections:

1. Establishes a connection to the Junos XML protocol server on the routing, switching, or security platform, as described in "Connecting to the Junos XML Protocol Server" on page 73.

2. Starts a Junos XML protocol session, as described in "Starting Junos XML Protocol Sessions" on page 75.

3. Optionally locks the candidate configuration, creates a private copy of the configuration, or opens an instance of the ephemeral configuration database.

   Locking the configuration prevents other users or applications from changing it at the same time. Creating a private copy of the configuration enables the application to make changes without affecting the candidate configuration until the copy is committed. For more information, see "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

   For information about the ephemeral configuration database, see "Understanding the Ephemeral Configuration Database" on page 310 and "Enabling and Configuring Instances of the Ephemeral Configuration Database" on page 328.

4. Makes any number of requests one at a time, freely intermingling operational and configuration requests. See "Requesting Operational Information Using the Junos XML Protocol" on page 359 and "Requesting Configuration Data Using the Junos XML Protocol" on page 375.

   The application can also intermix requests with configuration changes.

5. Accepts the tag stream emitted by the Junos XML protocol server in response to each request and extracts its content, as described in "Parsing the Junos XML Protocol Server Response" on page 87.

6. Unlocks the candidate configuration if it is locked or closes a private copy of the configuration or an open instance of the ephemeral configuration database.

   Other users and applications cannot change the candidate configuration while it remains locked. For more information, see "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

7. Ends the Junos XML protocol session and closes the connection to the device, as described in "Ending a Junos XML Protocol Session and Closing the Connection" on page 100.

RELATED DOCUMENTATION

- Request Operational Information Using the Junos XML Protocol | **359**
- Request Configuration Data Using the Junos XML Protocol | **375**
- Request the Complete Configuration Using the Junos XML Protocol | **414**
- Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | **94**

CHAPTER 5

Manage Junos XML Protocol Sessions

IN THIS CHAPTER

Satisfy the Prerequisites for Establishing a Connection to the Junos XML Protocol Server

IN THIS SECTION

A Junos XML protocol client application can choose between several protocols to connect to the Junos XML protocol server on devices running Junos OS or devices running Junos OS Evolved. To establish a connection to the server, a client application must satisfy the requirements that are applicable to all access protocols. A client application must also satisfy the requirements for the selected access protocol. The following sections outline the common and protocol-specific prerequisites.

Prerequisites for All Access Protocols

A client application must be able to log in to each device on which it establishes a connection with the Junos XML protocol server. You can create a Junos login account for the application, as described in this section. Alternatively, you can skip this section and enable authentication through RADIUS or TACACS +.

To create a local user account:

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

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

2. (Optional) Configure the `uid` and `full-name` statements to specify a unique user ID and the user's name.

```
[edit system login]
user@R0# set user netconf-user uid 2001 full-name "Junos XML protocol user"
```

3. Create a text-based password for the account. Include either the `plain-text-password` or `encrypted-password` statement at the `[edit system login user account-name authentication]` hierarchy level.

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

NOTE: A text-based password is not strictly necessary if the account accesses the Junos XML protocol server through a public/private keypair or certificate for authentication. However, we recommend that you create a password anyway. A password is required if you use the account for any other type of access, for example, to log in on the console. The password is also used if key-based or certificate-based authentication is configured but fails.

- To enter a password as text, issue the following command. The device prompts for the password and confirmation and then encrypts the password before storing it.

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

- To use a password that you previously created and hashed using MD5 or SHA1, issue the following command and provide the encrypted password:

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

4. Commit the configuration.

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

5. Repeat the preceding steps on each device where the client application establishes Junos XML protocol sessions.
6. Enable the client application to access the password and provide it when the Junos XML protocol server prompts for it. You can use several possible methods, including:
   - Code the application to prompt the user for a password at startup and to store the password temporarily in a secure manner.
   - Store the password in encrypted form in a secure local-disk location or secured database and code the application to access it.

Prerequisites for Clear-Text Connections

A client application can use the Junos XML protocol-specific clear-text access protocol to communicate with the Junos XML protocol server. The clear-text protocol sends unencrypted text directly over a TCP connection without using any additional protocol (such as SSH, SSL, or Telnet).

NOTE: Devices running the Junos-FIPS software do not accept Junos XML protocol clear-text connections. We recommend that you do not use the clear-text protocol in a Common Criteria environment. For more information, see the *Secure Configuration Guide for Common Criteria and Junos-FIPS*.

To enable client applications to use the clear-text protocol to connect to the Junos XML protocol server:

1. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.
2. Configure the `xnm-clear-text` statement at the `[edit system services]` hierarchy level to enable the Junos device to accept cleartext connections on port 3221.

   ```
   [edit system services]
   user@host# set xnm-clear-text
   ```

   For more information about the `xnm-clear-text` statement, see "Configure clear-text or SSL Service for Junos XML Protocol Client Applications" on page 71.
3. (Optional) Configure clear-text session options.

   Configure the `connection-limit` statement to limit the number of concurrent cleartext sessions; configure the `rate-limit` statement to limit the number of connection attempts. Both statements accept a value from 1 through 250.

   ```
   [edit system services]
   user@host# set xnm-clear-text connection-limit limit
   user@host# set xnm-clear-text rate-limit limit
   ```

   NOTE: By default, the Junos XML protocol server supports up to 75 simultaneous cleartext sessions and 150 connection attempts per minute.
4. Commit the configuration.

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

5. Repeat the steps on each device where the client application establishes Junos XML protocol sessions.

Prerequisites for SSH Connections

To enable a client application to use the SSH protocol to connect to the Junos XML protocol server, perform the following steps:

1. Enable the application to access the SSH software.

   Obtain the SSH software and install it on the computer where the application runs. For information about obtaining and installing SSH software, see http://www.ssh.com and http://www.openssh.com.
2. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.
3. (Optional) If you want to use key-based SSH authentication for the application, create a public/private keypair and associate it with the Junos OS login account you created in "Prerequisites for All Access Protocols" on page 61. Perform the following steps:

   a. Working on the computer where the client application runs, issue the `ssh-keygen` command in a standard command shell. For more information, see the man page provided by your SSH vendor for the `ssh-keygen` command.

   ```
   % ssh-keygen options
   ```

   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 (for example RSA) and SSH version.

   b. Enable the application to access the public and private keys. One method is to run the `ssh-agent` program on the computer where the application runs.

   c. On the Junos device, associate the public key with the login account.

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

   The command copies the contents of the specified file onto the Junos device. *URL* is the path to the file that contains one or more public keys.

   NOTE: Alternatively, you can add an RSA public key by configuring the `ssh-rsa` statement at the same hierarchy level and pasting in the public key.

4. Configure the `ssh` statement at the `[edit system services]` hierarchy level to enable the Junos device to accept SSH connections. This statement enables SSH access for all users and applications, not just Junos XML protocol client applications.

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

5. Commit the configuration.

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

6. Repeat Step 1 on each computer where the application runs. Repeat Step 2 through Step 5 on each device to which the application connects.

Prerequisites for Outbound SSH Connections

The outbound SSH feature allows the initiation of an SSH session between a Junos device and a network management system (NMS) where client-initiated TCP/IP connections are blocked (for example, when the device is behind a firewall). To enable outbound SSH, configure the `outbound-ssh` statement hierarchy on the Junos device. After you commit the configuration, the Junos device initiates outbound SSH sessions with the configured management clients. Once the outbound SSH session is initialized and the connection is established, the NMS initiates the SSH sequence as the client. The Junos device, acting as the server, authenticates the client.

Setting up outbound SSH involves:

- Configuring the device running Junos OS or the device running Junos OS Evolved for outbound SSH
- Configuring the management server for outbound SSH

To configure the Junos device for outbound SSH:

1. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.
2. Set the SSH protocol to v2.

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

3. Generate a public/private keypair for the Junos device. This keypair is used to encrypt the data transferred across the SSH connection.

   For more information about generating keypairs, see the User Access and Authentication Administration Guide for Junos OS.

4. To manually install the public key on the NMS, copy the public key to the NMS server.
5. Configure the `outbound-ssh` statement at the `[edit system services]` hierarchy level.

```
[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 (in-order | sticky);
        secret password;
        services  netconf;
    }
}
```

The options are as follows:

- *address*—(Required) Hostname or IPv4 or IPv6 address of the management server. You can list multiple clients by adding each client's IP address or hostname along with the following connection parameters.
  - `port` *port-number*—Define the outbound SSH port for the client. The default is port 22.
  - `retry` *number*–Specify the number of times the device attempts to establish an outbound SSH connection. The default is three tries.
  - `timeout` *seconds*—Specify the amount of time, in seconds, that the Junos device attempts to establish an outbound SSH connection. The default is 15 seconds.
- `client` *client-id*—(Required) Define an identifier for the `outbound-ssh` configuration stanza on the device. Each `outbound-ssh` stanza represents a single outbound SSH connection. The device does not send this attribute to the client.
- `device-id` *device-id*—(Required) Specify a name that identifies the Junos device to the client during the initiation sequence.

- keep-alive—(Optional) Specify that the device send keepalive messages to the management server. To configure the keepalive message, you must set both the timeout and retry attributes.
  - retry *number*—Specify the number of keepalive messages the device sends without receiving a response from the NMS before terminating the current SSH connection. The default is three tries.
  - timeout *seconds*—Specify the amount of time, in seconds, that the server waits for data before sending a keepalive signal. The default is 15 seconds.
- reconnect-strategy (in-order | sticky)—(Optional) Specify the method the router or switch uses to reestablish a disconnected outbound SSH connection. Two methods are available:
  - in-order—Specify that the network device first attempt to establish an outbound SSH session based on the management server address list. The device attempts to establish a session with the first server on the list. If this connection is not available, the device attempts to establish a session with the next server, and so on down the list until it establishes a connection.
  - sticky—Specify that the network device first attempt to reconnect to the management server to which it was last connected. If the connection is unavailable, the device attempts to establish a connection with the next client on the list and so on down the list until it establishes a connection.

  When reconnecting to a client, the device running Junos OS attempts to reconnect to the client based on the retry and timeout values for each of the clients listed in the configuration management server list.
- secret *password*—(Optional) Instruct the device to send its public SSH host key. During the initialization of the outbound SSH service, the router or switch passes its public key to the management server. This method is the recommended way of maintaining a current copy of the device's public key.
- services—(Required) Specify the services available for the session. Currently, NETCONF is the only service available.

6. Commit the configuration.

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

To set up the configuration management server:

1. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.
2. Enable the application to access the SSH software.

Obtain the SSH software and install it on the computer where the application runs. For information about obtaining and installing SSH software, see http://www.ssh.com and http://www.openssh.com.

3. (Optional) Manually install the device's public key for use with the SSH connection.

4. Configure the client system to receive and process initialization broadcast requests. The initialization requests use the following syntax:

   - (Recommended) If you configured the `secret` statement, the Junos device sends its public SSH key along with the initialization sequence. When the NMS receives the key, the client needs to determine what to do with the key. We recommend that you replace any current public SSH key for the device with the new key. This method ensures that the client always has the current key available for authentication.

     ```
     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
     ```

   - If you did not configure the `secret` statement, the Junos device does not send its public SSH key along with the initialization sequence. You need to manually install the current public SSH key for the device.

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

Prerequisites for SSL Connections

To enable a client application to use the SSL protocol to connect to the Junos XML protocol server, perform the following steps:

1. Enable the application to access the SSL software.

   Obtain the SSL software and install it on the computer where the application runs. For information about obtaining and installing the SSL software, see http://www.openssl.org.

2. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.

3. Use one of the following methods to obtain an authentication certificate in privacy-enhanced mail (PEM) format:

   - Request a certificate from a certificate authority (CA); these agencies usually charge a fee.

   - Generate a self-signed certificate.

For example, working on the computer where the client application runs, issue the following `openssl` command in a standard command shell.

```
% openssl req -x509 -nodes -newkey rsa:2048 -keyout certificate-file.pem -out certificate-file.pem
```

The command generates a self-signed certificate and an unencrypted 2048-bit RSA private key, and writes them to the file called ***certificate-file*.pem** in the working directory.

4. Import the certificate into the Junos configuration.

```
[edit security certificates local]
user@host# set certificate-name load-key-file URL-or-path
```

The *certificate-name* value is a unique identifier for the certificate.

The *URL-or-path* value specifies the file that contains the paired certificate and private key. Specify either the URL location on the client computer or a path on the local disk (if you copied the certificate file to the local device).

NOTE: The CLI expects the private key in the ***URL-or-path*** file to be unencrypted. If you encrypted the key, the CLI prompts you for the passphrase associated with it, decrypts it, and stores the unencrypted version.

5. Configure the `xnm-ssl` statement at the `[edit system services]` hierarchy level to enable the Junos device to accept SSL connections on port 3220.

   The *certificate-name* value is the unique name you assigned to the certificate in Step 4.

```
[edit system services]
user@host# set xnm-ssl local-certificate certificate-name
```

6. (Optional) Configure SSL session options.

   Configure the `connection-limit` statement to limit the number of concurrent sessions; configure the `rate-limit` statement to limit the number of connection attempts. Both statements accept a value from 1 through 250.

```
[edit system services]
user@host# set xnm-ssl connection-limit limit
user@host# set xnm-ssl rate-limit limit
```

> **NOTE**: By default, the Junos XML protocol server supports up to 75 simultaneous SSL sessions and 150 connection attempts per minute.

7. Commit the configuration.

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

8. Repeat Step 1 on each computer where the client application runs. Repeat Step 2 through Step 7 on each device to which the client application connects.

Prerequisites for Telnet Connections

To enable a client application to use the Telnet protocol to access the Junos XML protocol server, perform the steps described in this section.

> **NOTE**: Devices running the Junos-FIPS software do not accept Telnet connections. We recommend that you do not use the Telnet protocol in a Common Criteria environment. For more information, see the *Secure Configuration Guide for Common Criteria and Junos-FIPS*.

1. Verify that the application can access the Telnet software. On most operating systems, Telnet is accessible in the standard distribution.
2. Satisfy the prerequisites discussed in "Prerequisites for All Access Protocols" on page 61.
3. Configure the `telnet` statement at the `[edit system services]` hierarchy level. This statement enables Telnet access for all users and applications, not just Junos XML protocol client applications.

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

4. Repeat Step 1 on each computer where the application runs. Repeat Step 2 and Step 3 on each device to which the application connects.

RELATED DOCUMENTATION

Understanding the Client Application's Role in a Junos XML Protocol Session | 56

Supported Access Protocols for Junos XML Protocol Sessions | 55

Connect to the Junos XML Protocol Server | 73

Configure clear-text or SSL Service for Junos XML Protocol Client Applications

IN THIS SECTION

A Junos XML protocol client application can use one of four protocols to connect to the Junos XML protocol server on a router: clear-text (a Junos XML protocol-specific protocol for sending unencrypted text over a TCP connection), SSH, SSL, or Telnet. For clients to use the clear-text or SSL protocol, you must include Junos XML protocol-specific statements in the router configuration.

For more information, see the following topics:

Configuring clear-text Service for Junos XML Protocol Client Applications

To configure the router to accept clear-text connections from Junos XML protocol client applications on port 3221, include the `xnm-clear-text` statement at the `[edit system services]` hierarchy level:

```
[edit system services]
xnm-clear-text {
    connection-limit limit;
    rate-limit limit;
}
```

By default, the Junos XML protocol server supports a limited number of simultaneous clear-text sessions and connection attempts per minute. Optionally, you can include either or both of the following statements to change the defaults:

- `connection-limit` *limit*—Maximum number of simultaneous connections per protocol (IPv4 and IPv6) (a value from 1 through 250). The default is 75. When you configure a connection limit, the limit is applicable to the number of sessions per protocol (IPv4 and IPv6). For example, a connection limit of 10 allows 10 IPv6 clear-text service sessions and 10 IPv4 clear-text service sessions.

- rate-limit *limit*—Maximum number of connection attempts accepted per minute per protocol (IPv4 and IPv6). The range is a value from 1 through 250. The default is 150. When you configure a rate limit, the limit is applicable to the number of connection attempts per protocol (IPv4 and IPv6). For example, a rate limit of 10 allows 10 IPv6 session connection attempts per minute and 10 IPv4 session connection attempts per minute.

You cannot include the xnm-clear-text statement on routers that run the Junos-FIPS software. We recommend that you do not use the clear-text protocol in a Common Criteria environment.

Configuring SSL Service for Junos XML Protocol Client Applications

To configure the router to accept SSL connections from Junos XML protocol client applications on port 3220, include the xnm-ssl statement at the [edit system services] hierarchy level:

```
[edit system services]
xnm-ssl {
    local-certificate name;
    connection-limit limit;
    rate-limit limit;
}
```

local-certificate is the name of the X.509 authentication certificate used to establish an SSL connection. You must obtain the certificate and copy it to the router before referencing it.

By default, the Junos XML protocol server supports a limited number of simultaneous SSL sessions and connection attempts per minute. Optionally, you can include either or both of the following statements to change the defaults:

- connection-limit *limit*—Maximum number of simultaneous connections per protocol (IPV4 and IPv6). The range is a value from 1 through 250. The default is 75. When you configure a connection limit, the limit is applicable to the number of sessions per protocol (IPv4 and IPv6). For example, a connection limit of 10 allows 10 IPv6 SSL sessions and 10 IPv4 SSL sessions.

- rate-limit *limit*—Maximum number of connection attempts accepted per protocol per minute. The range is a value from 1 through 250. The default is 150. When you configure a rate limit, the limit is applicable to the number of connection attempts per protocol (IPv4 and IPv6). For example, a rate limit of 10 allows 10 IPv6 SSL session connection attempts per minute and 10 IPv4 SSL session connection attempts per minute.

Connect to the Junos XML Protocol Server

IN THIS SECTION

You can connect to the Junos XML protocol server through the Junos OS command-line interface (CLI) or through a client application. The following sections provide details for each method:

Connecting to the Junos XML Protocol Server from the CLI

The Junos XML management protocol and Junos XML API are primarily intended for use by client applications. However, for testing purposes you can establish an interactive Junos XML protocol session and type commands in a shell window.

To connect to the Junos XML protocol server from the CLI operational mode, issue the `junoscript interactive` command. The `interactive` option causes the Junos XML protocol server to echo what you type.

```
user@host> junoscript interactive
```

To begin a Junos XML protocol session over the connection, emit the initialization PI and tag that are described in "Start a Junos XML Protocol Session" on page 75. You can then enter tag element sequences that represent operational and configuration operations. To eliminate typing errors, save complete tag element sequences in a file and use a cut-and-paste utility to copy the sequences to the shell window.

NOTE: When you close the connection to the Junos XML protocol server (for example, by emitting the `<request-end-session/>` and `</junoscript>` tags), the device completely closes the connection instead of returning to the CLI operational mode prompt. For more information about ending a Junos XML protocol session, see "End a Junos XML Protocol Session and Close the Connection" on page 100.

Connecting to the Junos XML Protocol Server from the Client Application

For a client application to connect to the Junos XML protocol server and open a session, you must first satisfy the prerequisites described in "Satisfy the Prerequisites for Establishing a Connection to the Junos XML Protocol Server" on page 60.

A client application connects to the Junos XML protocol server by opening a socket or other communications channel to the Junos XML protocol server device, invoking one of the remote-connection routines appropriate for the programming language and access protocol that the application uses.

What the client application does next depends on which access protocol it is using:

- If using the clear-text or SSL protocol, the client application performs the following steps:

  1. Emits the initialization PI and tag, as described in "Start a Junos XML Protocol Session" on page 75.

  2. Authenticates with the Junos XML protocol server, as described in "Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections" on page 80.

- If using the SSH or Telnet protocol, the client application performs the following steps:

  1. Uses the protocol's built-in authentication mechanism to authenticate.

  2. Issues the `junoscript` command to request that the Junos XML protocol server convert the connection into a Junos XML protocol session.

     For a C programming language example, see "Establish a Junos XML Protocol Session Using C Client Applications" on page 459 and "Access and Edit Device Configurations Using Junos XML Protocol C Client Applications" on page 460.

  3. Emits the initialization PI and tag, as described in "Start a Junos XML Protocol Session" on page 75.

RELATED DOCUMENTATION

Understanding the Client Application's Role in a Junos XML Protocol Session | 56

Supported Access Protocols for Junos XML Protocol Sessions | 55

Start a Junos XML Protocol Session

IN THIS SECTION

Each Junos XML protocol session begins with a handshake in which the Junos XML protocol server and the client application specify the version of XML and the version of the Junos XML management protocol they are using. Each party parses the version information emitted by the other, using it to determine whether they can communicate successfully. Specifically, the client application emits an `<?xml?>` processing instruction (PI) and an opening `<junoscript>` tag. The following sections describe how to start a Junos XML protocol session.

Emitting the <?xml?> PI

The client application begins by emitting an `<?xml?>` PI.

```
<?xml version="version" encoding="encoding"?>
```

The attributes are as follows:

- version—The version of XML with which tag elements emitted by the client application comply
- encoding—The standardized character set that the client application uses and can understand

For a list of the attribute values that are acceptable in the current version of the Junos XML management protocol, see "Verifying Software Compatibility" on page 79.

In the following example of a client application's `<?xml?>` PI, the `version="1.0"` attribute indicates that the application is emitting tag elements that comply with the XML 1.0 specification. The `encoding="us-ascii"` attribute indicates that the client application is using the 7-bit ASCII character set standardized by the

American National Standards Institute (ANSI). For more information about ANSI standards, see http://www.ansi.org/ .

```
<?xml version="1.0" encoding="us-ascii"?>
```

NOTE: If the application fails to emit the <?xml?> PI before emitting the opening <junoscript> tag, the Junos XML protocol server emits an error message and immediately closes the session and connection.

Emitting the Opening <junoscript> Tag

The client application then emits its opening <junoscript> tag, which has the following syntax:

```
<junoscript version="version" hostname="hostname" junos:key="key" release="release-code">
```

The attributes are as follows. For a list of the attribute values that are acceptable in the current version of the Junos XML management protocol, see "Verifying Software Compatibility" on page 79.

version (Required) Specifies the version of the Junos XML management protocol that the client application is using.

hostname (Optional) Names the machine on which the client application is running. The information is used only when diagnosing problems. The Junos XML protocol does not include support for establishing trusted-host relationships or otherwise altering Junos XML protocol server behavior depending on the client hostname.

junos:key (Optional) Requests that the Junos XML protocol server indicate whether a child configuration element is an identifier for its parent element. The only acceptable value is "key". For more information, see "Requesting Identifier Indicators for Configuration Elements Using the Junos XML Protocol" on page 406.

release (Optional) Identifies the Junos OS Release (and by implication, the Junos XML API) for which the client application is designed. The value of this attribute indicates that the client application can interoperate successfully with a Junos XML protocol server that also supports that version of the Junos XML API. In other words, it indicates that the client application emits request tag elements from that API and knows how to parse response tag elements from it. If the application does not include this attribute, the Junos XML protocol server emits tag elements from the Junos XML API that it supports.

For the value of the release attribute, use the standard notation for Junos OS version numbers. For example, the value 20.4R1 represents the initial version of Junos OS Release 20.4.

In the following example of a client application's opening <junoscript> tag, the version="1.0" attribute indicates that it is using Junos XML protocol version 1.0. The hostname="client1" attribute indicates that the client application is running on the machine called client1. The release="20.4R1" attribute indicates that the network device is running the initial version of Junos OS Release 20.4.

```
<junoscript version="1.0" hostname="client1" release="20.4R1">
```

If the application fails to emit the <?xml?> PI before emitting the opening <junoscript> tag, the Junos XML protocol server emits an error message similar to the following and immediately closes the session and connection:

```
<rpc-reply>
    <xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm" \
        xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
        <message>
            communication error while exchanging credentials
        </message>
    </xnm:error>
</rpc-reply>
<!-- session end at YYYY-MM-DD hh:mm:ss TZ -->
</junoscript>
```

NOTE: For more information about the <xnm:error> tag, see "Handling an Error or Warning in Junos XML Protocol Sessions" on page 92.

Parsing the Junos XML Protocol Server's <?xml?> PI

When the Junos XML protocol session begins, the Junos XML protocol server emits an <?xml?> PI and an opening <junoscript> tag.

The syntax for the <?xml?> PI is as follows:

```
<?xml version="version" encoding="encoding"?>
```

The attributes are as follows. For a list of the attribute values that are acceptable in the current version of the Junos XML management protocol, see "Verifying Software Compatibility" on page 79.

version The version of XML with which tag elements emitted by the Junos XML protocol server comply

encoding The standardized character set that the Junos XML protocol server uses and can understand

In the following example of a Junos XML protocol server's <?xml?> PI, the version="1.0" attribute indicates that the server is emitting tag elements that comply with the XML 1.0 specification. The encoding="us-ascii" attribute indicates that the server is using the 7-bit ASCII character set standardized by ANSI.

```
<?xml version="1.0" encoding="us-ascii"?>
```

Parsing the Junos XML Protocol Server's Opening <junoscript> Tag

After emitting the <?xml?> PI, the server then emits its opening <junoscript> tag, which has the following form (the tag appears on multiple lines only for legibility):

```
<junoscript xmlns="namespace-URL" xmlns:junos="namespace-URL" \
  schemaLocation="namespace-URL" os="JUNOS" \
  release="release-code" hostname="hostname" version="version">
```

The attributes are as follows:

hostname The name of the device on which the Junos XML protocol server is running.

os The operating system of the device on which the Junos XML protocol server is running. The value is always JUNOS.

release The identifier for the version of the Junos OS from which the Junos XML protocol server is derived and that it is designed to understand. It is presumably in use on the device where the Junos XML protocol server is running. The value of the release attribute uses the standard notation for Juniper Networks software version numbers. For example, the value 20.4R1 represents the initial version of Junos OS Release 20.4.

schemaLocation The XML namespace for the XML Schema-language representation of the Junos OS configuration hierarchy.

version The version of the Junos XML management protocol that the Junos XML protocol server is using.

xmlns The XML namespace for the tag elements enclosed by the <junoscript> tag element that do not have a prefix on their names (that is, the default namespace for Junos XML tag elements). 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:junos The XML namespace for the tag elements enclosed by the <junoscript> tag element that have the junos: prefix on their names. The value is a URL of the form http://xml.juniper.net/junos/*release-code*/junos, where *release-code* is the standard string that represents a release of the Junos OS. For example, the value 20.4R1 represents the initial version of Junos OS Release 20.4.

In the following example of a Junos XML protocol server's opening <junoscript> tag, the version attribute indicates that the server is using Junos XML protocol version 1.0, and the hostname attribute indicates that the router's name is big-device. The os and release attributes indicate that the device is running the initial version of Junos OS Release 20.4. The xmlns attribute indicate that the default namespace for Junos XML tag elements is http://xml.juniper.net/xnm/1.1/xnm . The xmlns:junos attribute indicates that the namespace for tag elements that have the junos: prefix is http://xml.juniper.net/junos/20.4R1/junos . The tag appears on multiple lines only for legibility.

```
<junoscript xmlns="http://xml.juniper.net/xnm/1.1/xnm"\
     xmlns:junos="http://xml.juniper.net/junos/20.4R1/junos" \
    schemaLocation="http://xml.juniper.net/junos/20.4R1/junos" os="JUNOS" \
    release="20.4R1.8" hostname="big-device" version="1.0">
```

Verifying Software Compatibility

Exchanging the <?xml?> and <junoscript> elements enables a client application and the Junos XML protocol server to determine if they are running different versions of the software used during a Junos XML protocol session. Different versions are sometimes incompatible, and by Junos XML protocol convention the party running the later version of software determines how to handle any incompatibility. For fully automated performance, include code in the client application that determines if its version of software is later than that of the Junos XML protocol server. Decide which of the following options is appropriate when the application's version is more recent, and implement the corresponding response:

- Ignore differences in Junos version, and do not alter the client application's behavior to accommodate the Junos XML protocol server. A difference in Junos versions does not necessarily make the server and client incompatible, so this is often a valid approach.
- Alter standard behavior to be compatible with the Junos XML protocol server. If the client application is running a later version of the Junos OS, for example, it can choose to emit only tag elements that represent the software features available in the Junos XML protocol server's version of the Junos OS.
- End the Junos XML protocol session and terminate the connection. This is appropriate if you decide that it is not practical to accommodate the Junos XML protocol server's version of software. For instructions, see "Ending a Junos XML Protocol Session and Closing the Connection" on page 100.

Table 5 on page 80 specifies the PI or opening tag and attribute used to convey version information during Junos XML protocol session initialization in version 1.0 of the Junos XML management protocol.

Table 5: Junos XML Protocol version 1.0 PI and Opening Tag

Software and Versions	PI or Tag	Attribute
XML 1.0	`<?xml?>`	`version="1.0"`
ANSI-standardized 7-bit ASCII character set	`<?xml?>`	`encoding="us-ascii"`
Junos XML protocol 1.0	`<junoscript>`	`version="1.0"`

Junos OS Release	`<junoscript>`	`release="m.nZb"`
		For example:
		`release="10.3R1"`

RELATED DOCUMENTATION

- Understanding the Client Application's Role in a Junos XML Protocol Session | 56
- Sample Junos XML Protocol Session | 101

Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections

IN THIS SECTION

A Junos XML protocol client application that uses the cleartext or SSL protocol must authenticate with the Junos XML protocol server. (Applications that use the SSH or Telnet protocol use the protocol's built-in authentication mechanism.)

NOTE: The clear-text protocol is a Junos XML protocol-specific protocol for sending unencrypted text over a TCP connection. Because the protocol sends unencrypted text, thereby creating a potential security vulnerability, we recommend that you use SSH.

Submitting an Authentication Request

The client application begins the authentication process by emitting an `<rpc>` tag enclosing the `<request-login>` element. The `<request-login>` element encloses the `<username>` element to specify the Junos OS account (username) under which to establish the connection. You can choose whether the application provides the account password as part of the initial tag sequence.

NOTE: Any XML special characters in the username or password elements of a `<request-login>` RPC request must be escaped. Special characters include: greater than (>), less than (<), single quote ('), double quote ("), and ampersand (&). Both entity references and character references are acceptable escape sequence formats. For example, `&amp;` and `&#38;` are valid representations of an ampersand.

Providing the Username and Password

An application initially provides both the username and password in the following scenarios:

- The application automates access to Junos device information and does not interact with users.
- The application obtains the password from a user before beginning the authentication process.

To provide both the username and password, the application emits the following tag sequence:

```
<rpc>
    <request-login>
        <username>username</username>
        <challenge-response>password</challenge-response>
    </request-login>
</rpc>
```

Providing Only the Username

If the application instead obtains the password after the authentication process begins, the application initially specifies only the username.

To specify only the username and omit the password, the application emits the following tag sequence:

```
<rpc>
    <request-login>
        <username>username</username>
    </request-login>
</rpc>
```

In this case, the Junos XML protocol server returns an <rpc-reply> element with the <challenge> tag to request the password associated with the username. The element encloses the Password: string, which the client application can forward to the screen as a prompt for the user. The echo="no" attribute specifies that the password string typed by the user does not echo on the screen. The tag sequence is as follows:

```
<rpc-reply xmlns:junos="URL">
    <challenge echo="no">Password:</challenge>
</rpc-reply>
```

The client application obtains the password and emits the following tag sequence to forward it to the Junos XML protocol server:

```
<rpc>
    <request-login>
        <username>username</username>
        <challenge-response>password</challenge-response>
    </request-login>
</rpc>
```

Interpreting the Authentication Response

After it receives the username and password, the Junos XML protocol server emits the <authentication-response> element to indicate whether the authentication attempt is successful.

Server Response When Authentication Succeeds

If the password is correct, the authentication attempt succeeds and the Junos XML protocol server emits the following tag sequence:

```
<rpc-reply xmlns:junos="URL">
    <authentication-response>
        <status>success</status>
```

```
        <message>username</message>
        <login-name>remote-username</login-name>
    </authentication-response>
</rpc-reply>
```

The <authentication-response> child elements are:

- <status>—Status of the authentication request.
- <message>—The Junos username under which the connection is established.
- <login-name>—The username that the client application provided to an authentication utility such as RADIUS or TACACS+. This element appears only if the username differs from the username contained in the <message> element.

After successfully authenticating the user, the Junos XML protocol session begins, as described in "Starting Junos XML Protocol Sessions" on page 75.

Server Response When Authentication Fails

If the password is not correct or the <request-login> element is otherwise malformed, the authentication attempt fails and the Junos XML protocol server emits the following tag sequence:

```
<rpc-reply xmlns:junos="URL">
    <authentication-response>
        <status>fail</status>
        <message>error-message</message>
    </authentication-response>
</rpc-reply>
```

The *error-message* string in the <message> element explains why the authentication attempt failed. The Junos XML protocol server emits the <challenge> tag up to two more times before rejecting the authentication attempt and closing the connection.

RELATED DOCUMENTATION

Understanding the Client Application's Role in a Junos XML Protocol Session | 56

Supported Access Protocols for Junos XML Protocol Sessions | 55

Satisfy the Prerequisites for Establishing a Connection to the Junos XML Protocol Server | 60

Connect to the Junos XML Protocol Server | 73

<request-login> | 140

Send Requests to the Junos XML Protocol Server

IN THIS SECTION

In a Junos XML protocol session with a device running Junos OS, a client application initiates a request by emitting the opening <rpc> tag, 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 Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol server echoes the attribute in its opening <rpc-reply> tag, making it easy to map the response to the initiating request. The client application can freely define attribute names, except as described in the following note.

NOTE: The xmlns:junos attribute name is reserved. The Junos XML protocol server sets the attribute to an appropriate value on the opening <rpc-reply> tag, so client applications must not emit it in the opening <rpc> tag.

Although operational and configuration requests conceptually belong to separate classes, a Junos XML protocol session does not have distinct modes that correspond to CLI operational and configuration

modes. Each request tag 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

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 for many CLI commands. For example, the <get-interface-information> tag corresponds to the `show interfaces` command, and the <get-chassis-inventory> tag 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 about operational requests, see "Requesting Operational Information Using the Junos XML Protocol" on page 359. For information about the Junos XML request tag elements available in the current Junos OS Release, see the *Junos XML API Operational Developer Reference* and the XML API Explorer.

Configuration Information Requests

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

The Junos XML protocol defines the <get-configuration> 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 about the `[edit system login]` hierarchy level in the candidate configuration:

```
<rpc>
    <get-configuration>
        <configuration>
            <system>
```

```
                <login/>
            </system>
        </configuration>
    </get-configuration>
</rpc>
```

For more information about configuration information requests, see "Requesting Configuration Data Using the Junos XML Protocol" on page 375. For a summary of Junos XML configuration tag elements, see the *Junos XML API Configuration Developer Reference* and the XML API Explorer.

Configuration Change Requests

Configuration change requests are requests to change the configuration, or to commit those changes to put them into active use on the device running Junos OS. The Junos XML protocol defines the `<load-configuration>` operation 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>
    <load-configuration>
        <configuration>
            <system>
                <login>
                    <user>
                        <name>admin</name>
                        <full-name>Administrator</full-name>
                        <class>superuser</class>
                    </user>
                </login>
            </system>
        </configuration>
    </load-configuration>
</rpc>
```

For more information about configuration change requests, see "Requesting Configuration Changes Using the Junos XML Protocol" on page 206 and "Committing the Candidate Configuration Using the Junos XML Protocol" on page 288. For a summary of Junos XML configuration tag elements, see the *Junos XML API Configuration Developer Reference* and the XML API Explorer.

RELATED DOCUMENTATION

- Understanding the Client Application's Role in a Junos XML Protocol Session | 56
- XML and Junos XML Management Protocol Conventions Overview | 11
- Parse the Junos XML Protocol Server Response | 87
- Handle an Error or Warning in Junos XML Protocol Sessions | 92
- Halt a Request in Junos XML Protocol Sessions | 93

Parse the Junos XML Protocol Server Response

IN THIS SECTION

In a Junos XML protocol session with a device running Junos OS, a client application sends RPCs to the Junos XML protocol server to request information from and manage the configuration on the device. The Junos XML protocol 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:junos=""http://xml.juniper.net/junos/release/junos"">
    <!-- tag elements representing a response -->
</rpc-reply>
```

The xmlns:junos attribute in the opening <rpc-reply> tag defines the default namespace for the 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 Junos XML protocol server device, for example 20.4R1

The <rpc-reply> tag element occurs only within the <junoscript> element. Client applications must include code for parsing the stream of response tag elements coming from the Junos XML protocol server, either processing them as they arrive or storing them until the response is complete. The Junos XML protocol server returns three classes of responses:

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 Junos XML protocol server returns the information requested by the <get-interface-information> tag in a response tag called <interface-information>, and returns the information requested by the <get-chassis-inventory> tag in a<chassis-inventory> response tag called <chassis-inventory>. Operational responses also can be returned in formatted ASCII, which is enclosed within an output element, or in JSON format. For more information about formatting operational responses see "Specifying the Output Format for Operational Information Requests in a Junos XML Protocol Session" on page 363.

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 contains interface information for Junos OS Release 20.4.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.4R1/junos">
    <interface-information xmlns="http://xml.juniper.net/junos/20.4R1/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 contents of operational response tag elements, see "Requesting Operational Information Using the Junos XML Protocol" on page 359. 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. You can instruct the server to return configuration data in different formats including Junos XML elements, formatted ASCII, Junos OS set commands, or JSON. If you do not specify a format, the default is XML. For more information about formatting configuration information responses see "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382.

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.

```
<rpc-reply xmlns:junos="URL">
    <configuration>
        <system>
            <login>
                 <user>
                    <name>admin</name>
                    <full-name>Administrator</full-name>
                    <!-- other data tag elements for the admin user -->
                </user>
            </login>
        </system>
    </configuration>
</rpc-reply>
```

Configuration Change Responses

Configuration change responses are responses to requests that change the state or contents of the device configuration. For commit operations, the Junos XML protocol server returns the `<commit-results>` response tag, which encloses an explicit indicator of success or failure.

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <!-- tag elements for information about the commit -->
    </commit-results>
</rpc-reply>
```

For other operations, instead of emitting an explicit success indicator, the Junos XML protocol server indicates success by returning an opening `<rpc-reply>` tag and closing `</rpc-reply>` tag with no child elements.

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

For more information, see "Requesting Configuration Changes Using the Junos XML Protocol" on page 206 and "Committing the Candidate Configuration Using the Junos XML Protocol" on page 288. For a summary of the available configuration tag elements, see the *Junos XML API Configuration Developer Reference*.

RELATED DOCUMENTATION

Understanding the Client Application's Role in a Junos XML Protocol Session | 56

Send Requests to the Junos XML Protocol Server | 84

Handle an Error or Warning in Junos XML Protocol Sessions | 92

Parse Response Tag Elements Using a Standard API 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 https://metacpan.org/search?q=dist:XML-DOM+dom.

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/ .

How Character Encoding Works on Juniper Networks Devices

Junos OS configuration data and operational command output might contain non-ASCII characters, which are outside of the 7-bit ASCII character set. When displaying operational or configuration data in certain formats or within a certain type of session, the software escapes and encodes these characters. The software escapes or encodes the characters using the equivalent UTF-8 decimal character reference.

The CLI attempts to display any non-ASCII characters in configuration data that is produced in text, set, or JSON format. The CLI also attempts to display these characters in command output that is produced in text format. In the exception cases, the CLI displays the UTF-8 decimal character reference instead. (Exception cases include configuration data in XML format and command output in XML or JSON

format,) In NETCONF and Junos XML protocol sessions, you see a similar result if you request configuration data or command output that contains non-ASCII characters. In this case, the server returns the equivalent UTF-8 decimal character reference for those characters for all formats.

For example, suppose the following user account, which contains the Latin small letter n with a tilde (ñ), is configured on the device.

```
[edit]
user@host# set system login user mariap class super-user uid 2007 full-name "Maria Peña"
```

When you display the resulting configuration in text format, the CLI prints the corresponding character.

```
[edit]
user@host# show system login user mariap
full-name "Maria Peña";
uid 2007;
class super-user;
```

When you display the resulting configuration in XML format in the CLI, the ñ character maps to its equivalent UTF-8 decimal character reference Ã±. The same result occurs if you display the configuration in any format in a NETCONF or Junos XML protocol session.

```
[edit]
user@host# show system login user mariap | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.2R1/junos">
    <configuration junos:changed-seconds="1494033077" junos:changed-localtime="2017-05-05
18:11:17 PDT">
            <system>
                <login>
                    <user>
                        <name>mariap</name>
                        <full-name>Maria Pe&#195;&#177;a</full-name>
                        <uid>2007</uid>
                        <class>super-user</class>
                    </user>
                </login>
            </system>
    </configuration>
    <cli>
        <banner>[edit]</banner>
```

```
    </cli>
</rpc-reply>
```

When you load configuration data onto a device, you can load non-ASCII characters using their equivalent UTF-8 decimal character references.

Handle an Error or Warning in Junos XML Protocol Sessions

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

The syntax of the <xnm:error> element is as follows:

```
<xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm" \
     xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
    <!-- tag elements describing the error -->
</xnm:error>
```

The attributes are as follows:

- xmlns—The XML namespace for the <xnm:error> child tag elements that do not have a prefix in their names (that is, the default namespace for Junos XML tag elements). 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—The XML namespace for the <xnm:error> tag element and 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.

The set of child tags enclosed in the <xnm:error> element depends on the operation that server was performing when the error occurred. 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 a <load-configuration> tag element

Client applications must be prepared to receive and handle an <xnm:error> tag 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.

If the Junos XML protocol server encounters a less serious problem, it can emit an <xnm:warning> tag element instead. The usual response for the client application in this case is to log the warning or pass it to the user and to continue parsing the server's response.

RELATED DOCUMENTATION

- Send Requests to the Junos XML Protocol Server | 84
- Parse the Junos XML Protocol Server Response | 87
- Halt a Request in Junos XML Protocol Sessions | 93
- Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94
- <xnm:error> | 166
- <xnm:warning> | 169

Halt a Request in Junos XML Protocol Sessions

In a Junos XML protocol session, to request that the Junos XML protocol server stop processing the current request, a client application emits the <abort/> tag directly after the closing </rpc> tag for the operation to be halted.

```
<rpc>
    <!-- tag elements for the request -->
</rpc>
<abort/>
```

The Junos XML protocol server responds with the <abort-acknowledgement/> tag.

```
<rpc-reply xmlns:junos="URL">
    <abort-acknowledgement/>
</rpc-reply>
```

Depending on the operation being performed, response tag elements already sent by the Junos XML protocol server for the halted request are possibly invalid. The application can include logic for deciding whether to discard or retain them as appropriate.

RELATED DOCUMENTATION

- Send Requests to the Junos XML Protocol Server | 84
- Parse the Junos XML Protocol Server Response | 87
- Handle an Error or Warning in Junos XML Protocol Sessions | 92
- <abort/> | 113
- <abort-acknowledgement/> | 148

Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol

IN THIS SECTION

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

- Lock the candidate configuration, which prevents other users or applications from changing the shared configuration database until the application releases the lock (equivalent to the CLI `configure exclusive` command).
- Create a private copy of the candidate configuration, which enables the application to view or change configuration data without affecting the candidate or active configuration until the private copy is committed (equivalent to the CLI `configure private` 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 shared configuration database at the same time.

If an application is simply requesting configuration information and not changing it, locking the configuration or creating a private copy 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. The information from a private copy is guaranteed not to change, but can diverge from the candidate configuration if other users or applications are changing the candidate configuration.

The restrictions on, and interactions between, operations on the locked regular candidate configuration and a private copy are the same as for the CLI `configure exclusive` and `configure private` commands. For more information, see "Committing a Private Copy of the Configuration Using the Junos XML Protocol" on page 290 and the CLI User Guide.

For more information about locking and unlocking the candidate configuration or creating a private copy, see the following sections:

Locking the Candidate Configuration

To lock the candidate configuration, a client application emits the `<lock-configuration/>` tag within an `<rpc>` tag.

```
<rpc>
    <lock-configuration/>
</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 Junos XML protocol confirms that it has locked the candidate configuration by returning an opening `<rpc-reply>` and closing `</rpc-reply>` tag with nothing between them.

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

If the Junos XML protocol server cannot lock the configuration, the `<rpc-reply>` tag instead encloses an `<xnm:error>` 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 login identity of the user or application.

- The candidate configuration already includes changes that have not yet been committed. To commit the changes, see "Committing the Candidate Configuration Using the Junos XML Protocol" on page 288. To discard uncommitted changes, see "Replacing the Configuration Using the Junos XML Protocol" on page 219.

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, or can change their private copies. The lock persists until either the Junos XML protocol session ends or the client application unlocks the configuration by emitting the <unlock-configuration/> tag, as described in "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

If the candidate configuration is not committed before the client application unlocks it, or if the Junos XML protocol 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-configuration/> tag in an <rpc> tag:

```
<rpc>
    <unlock-configuration/>
</rpc>
```

The Junos XML protocol server confirms that it has successfully unlocked the configuration by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them.

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

If the Junos XML protocol server cannot unlock the configuration, the <rpc-reply> tag instead encloses an <xnm:error> element explaining the reason for the failure.

Creating a Private Copy of the Configuration

To create a private copy of the candidate configuration, a client application emits the <private/> tag enclosed in <rpc> and <open-configuration> tags.

```
<rpc>
    <open-configuration>
        <private/>
```

```
    </open-configuration>
</rpc>
```

The client application can then perform the same operations on the private copy as on the regular candidate configuration.

After making changes to the private copy, the client application can commit the changes to the active configuration on the device running Junos OS by emitting the <commit-configuration> tag element, as for the regular candidate configuration. However, there are some restrictions on the commit operation for a private copy. For more information, see "Committing a Private Copy of the Configuration Using the Junos XML Protocol" on page 290.

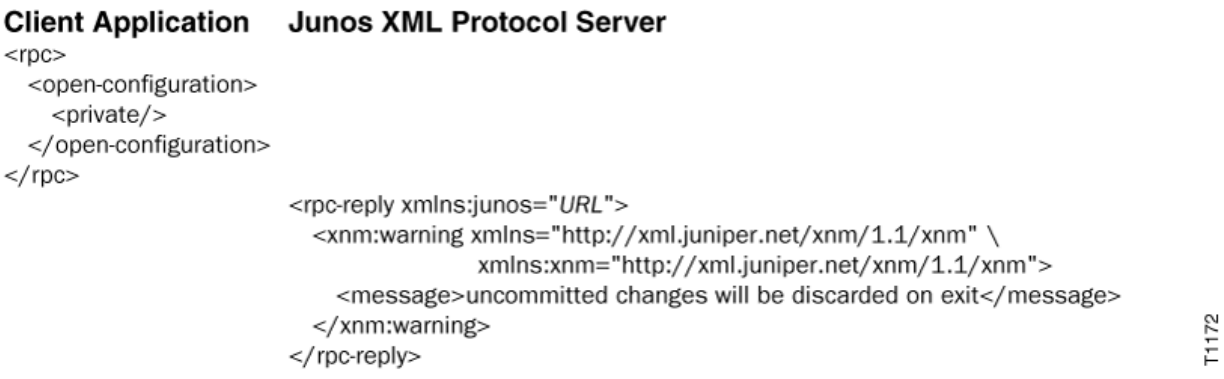
To discard the private copy without committing it, a client application emits the <close-configuration/> tag enclosed in an <rpc> tag element.

```
<rpc>
    <close-configuration/>
</rpc>
```

Any changes to the private copy are lost. Changes to the private copy are also lost if the 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 element.

NOTE: Starting in Junos OS Release 18.2R1, the Junos XML protocol <open-configuration> operation does not emit an "uncommitted changes will be discarded on exit" warning message when opening a private copy of the candidate configuration. However, Junos OS still discards the uncommitted changes upon closing the private copy.

The following example shows how to create a private copy of the configuration. The Junos XML protocol server includes a reminder in its confirmation response that changes are discarded from a private copy if they are not committed before the session ends.



Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
18.2R1	Starting in Junos OS Release 18.2R1, the Junos XML protocol <open-configuration> operation does not emit an "uncommitted changes will be discarded on exit" warning message when opening a private copy of the candidate configuration.

RELATED DOCUMENTATION

- Understanding the Client Application's Role in a Junos XML Protocol Session | 56
- Send Requests to the Junos XML Protocol Server | 84
- Request Configuration Changes Using the Junos XML Protocol | 206
- <lock-configuration/> | 136
- <unlock-configuration/> | 143
- <open-configuration> | 137

Terminate a Junos XML Protocol Session

In a Junos XML protocol 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 Junos XML protocol 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">
    <xnm:error>
        <message>
            configuration database locked by:
user terminal (pid PID) on since YYYY-MM-DD hh:mm:ss TZ, idle hh:mm:ss
            exclusive [edit]
        </message>
    </xnm: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>` 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 Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol 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 the Junos XML Protocol" on page 294.

RELATED DOCUMENTATION

- Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94
- Send Requests to the Junos XML Protocol Server | 84
- End a Junos XML Protocol Session and Close the Connection | 100
- Handle an Error or Warning in Junos XML Protocol Sessions | 92
- <kill-session> | 129

End a Junos XML Protocol Session and Close the Connection

In a Junos XML protocol session with a Junos device, when a client application is finished making requests, it ends the session by emitting the <request-end-session/> tag within an <rpc> tag element.

```
<rpc>
    <request-end-session/>
</rpc>
```

In response, the Junos XML protocol server emits the <end-session/> tag enclosed in an <rpc-reply> tag element and a closing </junoscript> tag.

```
<rpc-reply xmlns:junos="URL">
    <end-session/>
</rpc-reply>
</junoscript>
```

The client application waits to receive this reply before emitting its closing </junoscript> tag.

```
</junoscript>
```

The client application can then close the SSH, SSL, or other connection to the Junos XML protocol server device.

RELATED DOCUMENTATION

- Understanding the Client Application's Role in a Junos XML Protocol Session | 56
- Start a Junos XML Protocol Session | 75
- Terminate a Junos XML Protocol Session | 98
- <request-end-session/> | 139

Sample Junos XML Protocol Session

IN THIS SECTION

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

Exchanging Initialization PIs and Tag Elements

After the client application establishes a connection to a Junos XML protocol server, the two exchange initialization PIs and tag elements, as shown in the following example. Note that the Junos XML protocol server's opening <junoscript> tag appears on multiple lines for legibility only. Neither the Junos XML protocol server nor the client application inserts a newline character into the list of attributes. Also, in an actual exchange, the *JUNOS-release* variable is replaced by a value such as 20.4R1 for Junos OS Release 20.4. For a detailed discussion of the <?xml?> PI and opening <junoscript> tag, see "Starting Junos XML Protocol Sessions" on page 75.

Client Application

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" release="JUNOS-release">
```

Junos XML Protocol Server

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" hostname="router1" \
  os="JUNOS" release="JUNOS-release" \
  xmlns="URL"xmlns:junos="URL" \
  xmlns:xnm="URL">
```

T1173

Sending an Operational Request

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

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-chassis-inventory>
    <detail/>
  </get-chassis-inventory>
</rpc>
```

```
<rpc-reply 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>
```

T1102

Locking the Configuration

The client application then prepares to then prepares to incorporate a change into the candidate configuration by emitting the `<lock-configuration/>` 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 Junos XML protocol server returns only an opening `<rpc-reply>` tag and a closing `</rpc-reply>` tag with no child elements. For more information about locking the configuration, see "Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <lock-configuration/>
</rpc>
```

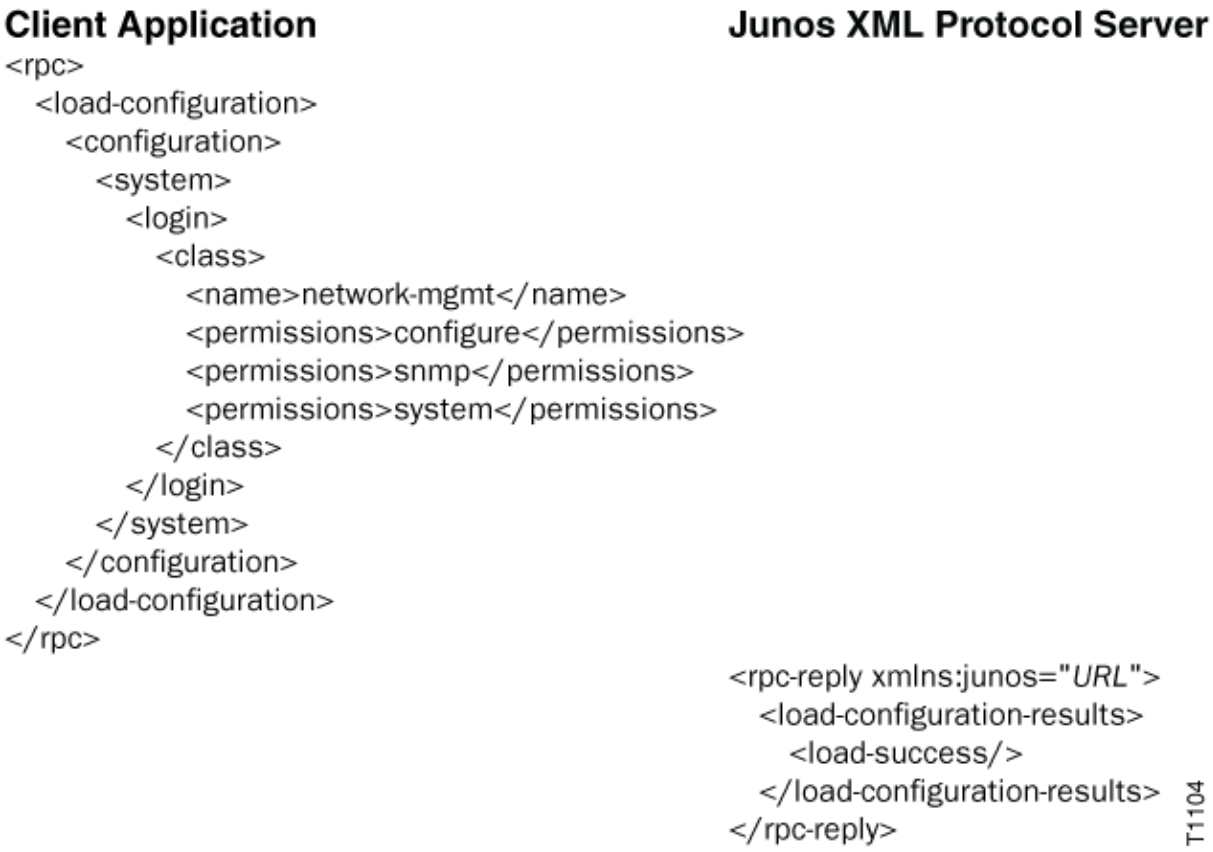
```
<rpc-reply xmlns:junos="URL">
</rpc-reply>
```

T1103

Changing the Configuration

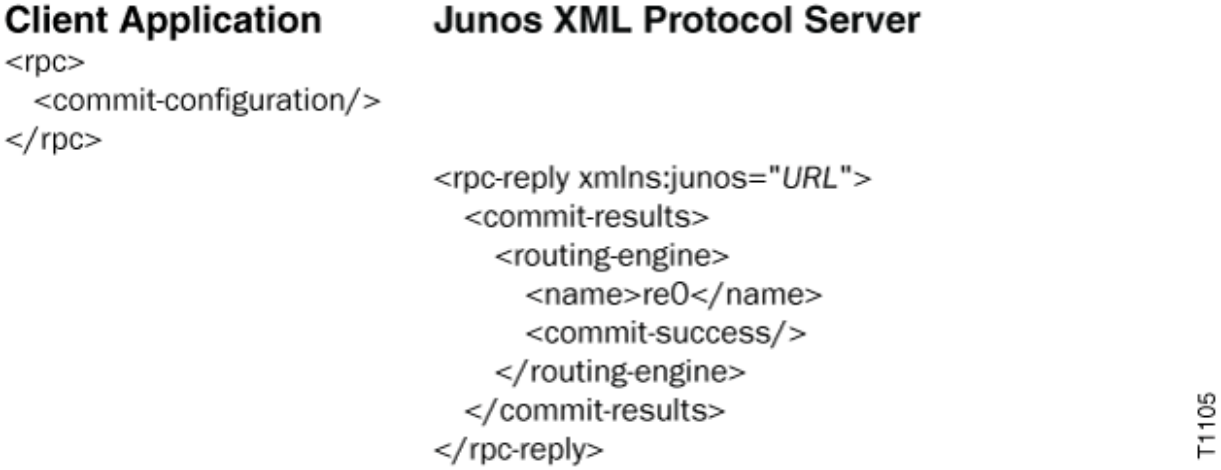
The client application now emits tag elements to create a new Junos OS login class called `network-mgmt` at the `[edit system login class]` hierarchy level in the candidate configuration. The Junos XML protocol server returns the `<load-configuration-results>` tag, which encloses a child element that reports the

outcome of the load operation. (Understanding the meaning of these tag elements is not necessary for the purposes of this example, but for information about them, see "Requesting Configuration Changes Using the Junos XML Protocol" on page 206.)



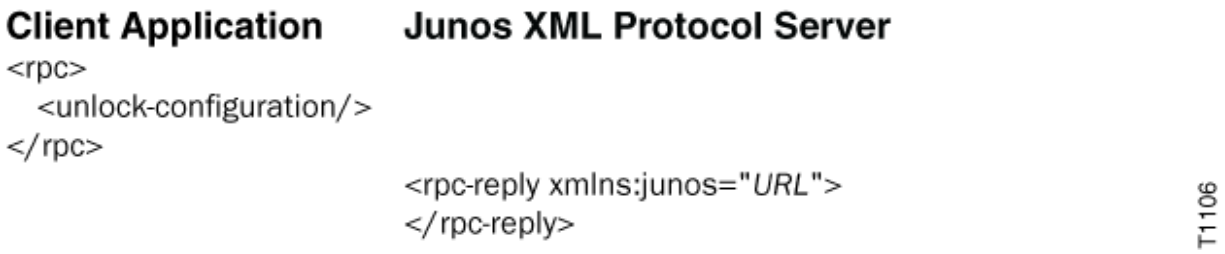
Committing the Configuration

The client application then commits the candidate configuration. The Junos XML protocol server returns the `<commit-results>` tag, which encloses child elements that report the outcome of the commit operation.



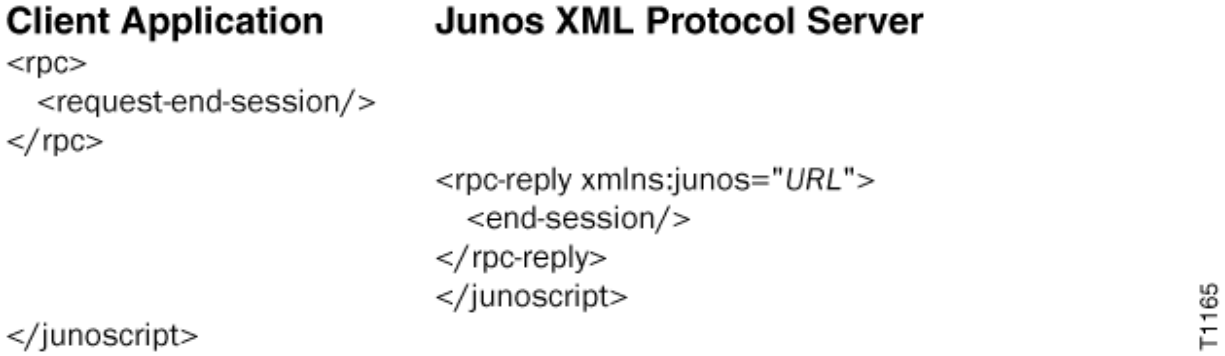
Unlocking the Configuration

The client application unlocks (and by implication closes) the candidate configuration. To confirm that the unlock operation was successful, the Junos XML protocol server returns only an opening `<rpc-reply>` tag and a closing `</rpc-reply>` tag with no child elements.



Closing the Junos XML Protocol Session

The client application closes the Junos XML protocol session by emitting the `<request-end-session>` tag.



RELATED DOCUMENTATION

- Understanding the Client Application's Role in a Junos XML Protocol Session | 56
- Connect to the Junos XML Protocol Server | 73
- Start a Junos XML Protocol Session | 75
- Send Requests to the Junos XML Protocol Server | 84
- Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94
- End a Junos XML Protocol Session and Close the Connection | 100

CHAPTER 6

Junos XML Protocol Tracing Operations

IN THIS CHAPTER

NETCONF and Junos XML Protocol Tracing Operations Overview

You can configure tracing operations for the NETCONF and Junos XML management protocols. NETCONF and Junos XML protocol tracing operations record NETCONF and Junos XML protocol session data, respectively, in a trace file. By default, devices running Junos OS and devices running Junos OS Evolved do not enable NETCONF or Junos XML protocol tracing operations.

You configure NETCONF and Junos XML protocol tracing operations at the `[edit system services netconf traceoptions]` hierarchy level. When you enable tracing operations, the configuration applies to both NETCONF and Junos XML protocol sessions. The system adds the `[NETCONF]` or `[JUNOScript]` tag to the log file entries to distinguish the session type.

```
[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 tracing operations and trace all incoming and outgoing data from NETCONF and Junos XML protocol sessions on a device, configure the `flag all` statement. You can also configure the `flag debug` statement to enable debug-level tracing. However, we recommend using the `flag all` option.

You can restrict tracing to only incoming or outgoing session data by configuring the flag value as either `incoming` or `outgoing`, respectively. Additionally, you can restrict the trace output to include only those lines that match a particular expression. To use specific match criteria, configure the `match` statement and define the regular expression against which to match the output.

To control the tracing operation from within a NETCONF or Junos XML protocol session, configure the `on-demand` statement. This option requires that you start and stop trace operations from within the session. To start tracing for that session, issue the following RPC within the session:

```
<rpc><request-netconf-trace><start/></request-netconf-trace></rpc>
```

To stop tracing for that session, issue the following RPC:

```
<rpc><request-netconf-trace><stop/></request-netconf-trace></rpc>
```

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

By default, when the trace file reaches 128 KB in size, it is compressed and renamed 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 *`size`*`k` to specify KB, *`size`*`m` to specify MB, or *`size`*`g` to specify GB. You cannot configure the maximum number of trace files and the maximum trace file size independently. If you configure one option, you must also configure the other option along with a filename.

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

RELATED DOCUMENTATION

Example: Trace NETCONF and Junos XML Protocol Session Operations | **107**

netconf

ssh (NETCONF)

traceoptions (NETCONF and Junos XML Protocol)

Example: Trace NETCONF and Junos XML Protocol Session Operations

IN THIS SECTION

This example configures tracing operations for NETCONF and Junos XML protocol sessions.

Requirements

- A device running Junos OS or a device running Junos OS Evolved.

Overview

This example configures basic tracing operations for NETCONF and Junos XML protocol sessions. When you configure tracing operations at the `[edit system services netconf traceoptions]` hierarchy, the device enables tracing operations for both NETCONF and Junos XML protocol sessions. The system adds the `[NETCONF]` or `[JUNOScript]` tag to the log file entries to distinguish the session type.

In this example, you configure the trace file **netconf-ops.log**. You configure 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 and Junos XML protocol data. The `world-readable` option enables unrestricted access to the trace files.

Configuration

IN THIS SECTION

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
```

Configure NETCONF and Junos XML Protocol Tracing Operations

Step-by-Step Procedure

To configure NETCONF and Junos XML protocol tracing operations:

1. For NETCONF sessions, enable NETCONF over SSH.

   ```
   [edit]
   user@R1# set system services netconf ssh
   ```

2. Configure the traceoptions flag to specify which session data to capture.

   You can specify incoming, outgoing, all, or debug data. This example configures tracing for all 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 **/var/log/netconf-ops.log**. If you do not specify a filename, the system logs NETCONF and Junos XML protocol session data in **/var/log/netconf**.

   ```
   [edit]
   user@R1# set system services netconf traceoptions file netconf-ops.log
   ```

4. (Optional) Configure the maximum number of 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 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 or Junos XML protocol 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

IN THIS SECTION

Verify NETCONF and Junos XML Protocol Tracing Operation

Purpose

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

Action

Display the configured trace file by issuing the **show log *filename*** operational mode command.

```
user@R1 show log netconf-ops.log
Apr  3 13:09:04 [NETCONF] Started tracing session: 3694
Apr  3 13:09:29 [NETCONF] - [3694] Incoming: <rpc>
```

```
Apr  3 13:09:29 [NETCONF] - [3694] Outgoing: <rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="http://xml.juniper.net/junos/24.4R1/
junos">
Apr  3 13:09:39 [NETCONF] - [3694] Incoming: <edit-config>
Apr  3 13:09:43 [NETCONF] - [3694] Incoming: <target>
Apr  3 13:09:47 [NETCONF] - [3694] Incoming: <candidate/>
Apr  3 13:09:53 [NETCONF] - [3694] Incoming: </target>
Apr  3 13:10:07 [NETCONF] - [3694] Incoming: <default-operation>merge</default-operation>
Apr  3 13:10:10 [NETCONF] - [3694] Incoming: <config>
Apr  3 13:10:13 [NETCONF] - [3694] Incoming: <configuration>
Apr  3 13:10:16 [NETCONF] - [3694] Incoming: <system>
Apr  3 13:10:19 [NETCONF] - [3694] Incoming: <scripts>
Apr  3 13:10:23 [NETCONF] - [3694] Incoming: <op>
Apr  3 13:10:26 [NETCONF] - [3694] Incoming: <file>
Apr  3 13:10:44 [NETCONF] - [3694] Incoming: <name>bgp-troubleshoot.slax</name>
Apr  3 13:10:46 [NETCONF] - [3694] Incoming: </file>
Apr  3 13:10:48 [NETCONF] - [3694] Incoming: </op>
Apr  3 13:10:52 [NETCONF] - [3694] Incoming: </scripts>
Apr  3 13:10:56 [NETCONF] - [3694] Incoming: </system>
Apr  3 13:11:00 [NETCONF] - [3694] Incoming: </configuration>
Apr  3 13:11:00 [NETCONF] - [3694] Outgoing: <ok/>
Apr  3 13:11:12 [NETCONF] - [3694] Incoming: </config>
Apr  3 13:11:18 [NETCONF] - [3694] Incoming: </edit-config>
Apr  3 13:11:26 [NETCONF] - [3694] Incoming: </rpc>
Apr  3 13:11:26 [NETCONF] - [3694] Outgoing: </rpc-reply>
Apr  3 13:11:26 [NETCONF] - [3694] Outgoing: ]]>]]>
Apr  3 13:11:31 [NETCONF] - [3694] Incoming: ]]>]]>

Apr  3 13:14:20 [NETCONF] - [3694] Incoming: <rpc>
Apr  3 13:14:20 [NETCONF] - [3694] Outgoing: <rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="http://xml.juniper.net/junos/24.4R1/
junos">
Apr  3 13:14:26 [NETCONF] - [3694] Incoming: <commit/>
Apr  3 13:14:35 [NETCONF] - [3694] Outgoing: <ok/>
Apr  3 13:14:35 [NETCONF] - [3694] Incoming: </rpc>
Apr  3 13:14:35 [NETCONF] - [3694] Outgoing: </rpc-reply>
Apr  3 13:14:35 [NETCONF] - [3694] Outgoing: ]]>]]>
Apr  3 13:14:40 [NETCONF] - [3694] Incoming: ]]>]]>

Apr  3 13:30:48 [NETCONF] - [3694] Outgoing: <!-- session end at 2025-04-03 13:30:48 PDT -->
```

Meaning

This example configures the `flag all` statement, so the trace file logs all incoming and outgoing data for any NETCONF and Junos XML protocol sessions. Each operation includes the date and timestamp. The log file indicates the session type, either NETCONF or Junos XML protocol, by including the `[NETCONF]` or `[JUNOScript]` tag, respectively. The device distinguishes multiple NETCONF and Junos XML protocol sessions by using a unique session number. In this example, only one NETCONF session, using session identifier 3694, is active.

RELATED DOCUMENTATION

NETCONF and Junos XML Protocol Tracing Operations Overview | **105**

traceoptions (NETCONF and Junos XML Protocol)

CHAPTER 7

Junos XML Protocol Operations

IN THIS CHAPTER

<abort/>

IN THIS SECTION

Usage

```
<rpc>
    <!-- child tag elements -->
</rpc>
<abort/>
```

Description

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. If the server already sent tagged data in response to the request, the client application must discard those elements.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

Halt a Request in Junos XML Protocol Sessions | 93

<abort-acknowledgement/> | 148

<close-configuration/>

IN THIS SECTION

Usage

```
<rpc>
    <close-configuration/>
</rpc>
```

Description

Close the open configuration database and discard any uncommitted changes.

This operation is normally used to close a private copy of the candidate configuration or an open instance of the ephemeral configuration database and discard any uncommitted changes. The application must have previously emitted the <open-configuration> operation. Closing the NETCONF or Junos XML protocol session (by emitting the <request-end-session/> tag, for example) has the same effect as emitting this operation.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94

<open-configuration> | 137

<request-end-session/> | 139

<commit-configuration>

IN THIS SECTION

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>
```

Description

Request that the NETCONF or Junos XML protocol server perform one of the variants of the commit operation. You can perform the commit operation on the candidate configuration, a private copy of the candidate configuration, or an open instance of the ephemeral configuration database.

On devices with dual Routing Engines, you can commit the candidate configuration, private copy, or ephemeral database instance stored on the local Routing Engine on both Routing Engines. The ephemeral database supports only the `<synchronize/>` option.

Some restrictions apply to the commit operation for a private copy of the candidate configuration and for the ephemeral configuration database. For example:

- The commit operation fails for a private copy 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.
- A commit operation on an instance of the ephemeral configuration database supports only the `<synchronize/>` option.
- The confirmed commit operation is not available when committing a private copy of the configuration or an open instance of the ephemeral configuration database.

To execute a commit or commit synchronize operation, enclose the appropriate tags in the <commit-configuration> tag element to specify the type of commit operation. Table 6 on page 118 and Table 7 on page 119 describe common commit and commit synchronize operations.

Table 6: Commit Operations

<commit-configuration> Operation	Description
<commit-configuration/>	Commit the configuration immediately, making it the active configuration on the device.
<commit-configuration> <check/> </commit-configuration>	Verify the syntactic correctness of the candidate configuration or a private copy without actually committing it.
<commit-configuration> <confirmed/> </commit-configuration> <commit-configuration> <confirmed/> <confirm-timeout>*rollback-delay*</confirm-timeout> </commit-configuration>	Commit the candidate configuration but require an explicit confirmation for the commit to become permanent. If the commit is not confirmed, the configuration rolls back to the previous configuration after the specified time. Optionally include the <confirm-timeout> element to specify the rollback delay in the range from 1 through 65,535 minutes. By default, the rollback occurs after 10 minutes. To delay the rollback again (past the original rollback deadline), emit the <commit-configuration><confirmed/></commit-configuration> tags before the deadline passes, and optionally Include the <confirm-timeout> element. The rollback can be delayed repeatedly in this way. To confirm the commit, emit the empty <commit-configuration/> tag or the <commit-configuration><check/><commit-configuration> tags before the rollback deadline passes. The device commits the candidate configuration and cancels the rollback.

Table 7: Commit Synchronize Operations

<commit-configuration> Operation	Description
<commit-configuration> <synchronize/> </commit-configuration>	Copy the candidate configuration or the open ephemeral instance data from the local Routing Engine to the other Routing Engine, verify the configuration's syntactic correctness, and commit it immediately on both Routing Engines.
<commit-configuration> <synchronize/> <at-time>*time-specification*</at-time> </commit-configuration>	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. You can also specify <force-synchronize/>.
<commit-configuration> <synchronize/> <check/> </commit-configuration>	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. You can also specify <force-synchronize/>.
<commit-configuration> <synchronize/> <confirmed/> <confirm-timeout>*rollback-delay*</confirm-timeout> </commit-configuration>	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.
<commit-configuration> <synchronize/> <force-synchronize/> </commit-configuration>	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.

To schedule the candidate configuration for commit at a future time, enclose the <at-time> element in the <commit-configuration> element. When you execute the operation, 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. Table 8 on page 120 outlines the valid types of time specifiers.

Table 8: <at-time> Time Specifiers

Time Specifier	Description	Example
reboot	Commit the configuration the next time the device reboots.	• <at-time>reboot</at-time>
hh:*mm*[:*ss*]	Commit the configuration at the specified time (hours, minutes, and, optionally, seconds). The time must be in the future but before 11:59:59 PM on the current day. Use 24-hour time for the *hh* value. The device interprets the time with respect to its clock and time zone settings.	• Execute the operation at 4:30:00 AM: <at-time>04:30:00</at-time> • Execute the operation at 8:00 PM: <at-time>20:00</at-time>
yyyy-mm-dd hh:*mm*[:*ss*]	Commit the configuration at the specified date and time (year, month, date, hours, minutes, and, optionally, seconds). The specified time must be after you execute the <commit-configuration> operation. Use 24-hour time for the *hh* value. The device interprets the time with respect to its clock and time zone settings.	• Execute the operation at 3:30 PM on August 21, 2005: <at-time>2005-08-21 15:30:00</at-time>

Contents

<at-time> Schedule the commit operation for a specified future time. Valid time specifiers include:

- reboot
- *hh*:*mm*[:*ss*]
- *yyyy-mm-dd hh*:*mm*[:*ss*]

<check> Request verification that the configuration is syntactically correct, but do not actually commit it.

<confirmed> Request a commit of the candidate configuration and require an explicit confirmation for the commit to become permanent. If the commit is not confirmed, roll back 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>	Specify the number of minutes for which the configuration remains active when the <confirmed/> tag is enclosed in the <commit-configuration> tag element. • **Range:** 1 through 65,535 minutes • **Default:** 10 minutes
<log>	Record a message in the commit history log when the commit operation succeeds.
<synchronize>	On dual control plane systems, request that the 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, force the candidate configuration on one control plane to be copied to the other control plane.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

- Commit the Candidate Configuration Using the Junos XML Protocol | **288**
- Commit a Private Copy of the Configuration Using the Junos XML Protocol | **290**
- Commit a Configuration at a Specified Time Using the Junos XML Protocol | **292**
- Commit the Candidate Configuration Only After Confirmation Using the Junos XML Protocol | **294**
- Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | **298**
- <commit-results> | **153**

<get-checksum-information>

IN THIS SECTION

Usage

```
<rpc>
    <get-checksum-information>
        <path>
        <!-- name and path of file -->
        </path>
    </get-checksum-information>
</rpc>
```

Description

Request checksum information for the specified file.

Contents

<path>	Name and path of the file to check.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

Operation added in Junos OS Release 9.2R1.

RELATED DOCUMENTATION

<checksum-information> | 152

<get-configuration>

IN THIS SECTION

Usage

```
<rpc>
    <get-configuration
        [changed="changed"]
        [commit-scripts="( apply | apply-no-transients | view )"]
        [compare=("configuration-revision" [configuration-revision="revision-id"] | "rollback"
[rollback="[0-49]"])]
        [database="(candidate | committed)"]
        [database-path=$junos-context/commit-context/database-path]
        [format="( json | set | 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>
```

Description

Request configuration data from the NETCONF or Junos XML protocol server. The attributes specify the source and formatting of the data to display.

If a client application issues the Junos XML protocol <open-configuration> operation to open a specific configuration database before executing the <get-configuration> operation, the server returns the configuration data from the open configuration database. Otherwise, the server returns the configuration data from the candidate configuration. You can explicitly request the active configuration database by including the database="committed" attribute.

A client application can request the entire configuration hierarchy or a subset of it.

- Entire configuration hierarchy—To display the entire configuration hierarchy, emit the empty <get-configuration/> tag.
- Subset of configuration hierarchy—To display a configuration element (hierarchy level or configuration object), emit the <get-configuration> element and include the elements that represent all levels of the configuration hierarchy from the root (<configuration>) 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: To retrieve configuration data from an instance of the ephemeral configuration database, a client application must first open the ephemeral instance using the <open-configuration> operation with the appropriate child tags before emitting the <get-configuration> operation. When retrieving ephemeral configuration data using the <get-configuration> operation, the only supported attributes are format and key.

NOTE: You can use the <get-configuration> operation to request the entire logical system configuration or request specific logical system configuration hierarchies using child configuration tags.

Attributes

changed Request that the junos:changed="changed" attribute appear in the opening tag of each changed configuration element.

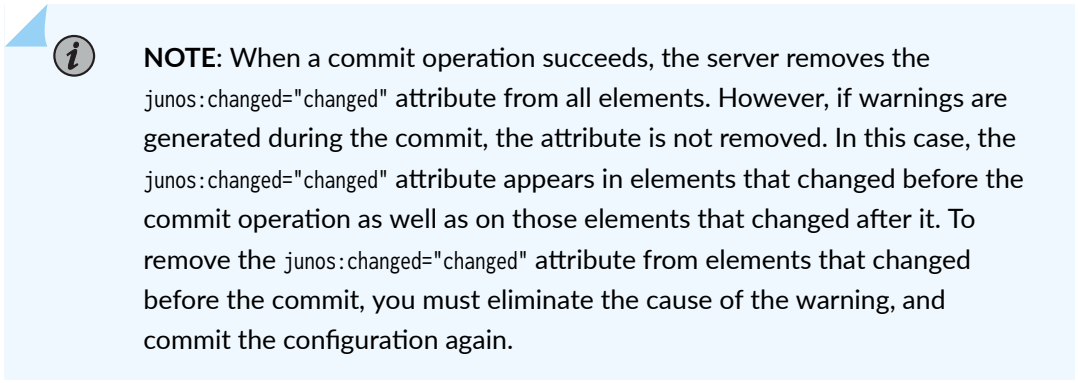
The attribute appears in the opening tag of every parent tag in the path to the changed configuration element, including the top-level <configuration> tag. If the changed configuration element is represented by a single (empty) tag, the junos:changed="changed"

attribute appears in the tag. If the changed element is represented by a container tag, the `junos:changed="changed"` attribute appears in the opening container tag and also in each child tag enclosed in the container tag element.

- **Values:** `changed`

The `database` attribute can be combined with the `changed="changed"` attribute to request either the candidate or active configuration:

- When you request the candidate configuration, elements added to the candidate configuration after the last commit operation are marked with the `junos:changed="changed"` attribute.
- When you request the active configuration, 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 elements. However, if warnings are generated during the commit, the attribute is not removed. In this case, the `junos:changed="changed"` attribute appears in elements that changed before the commit operation as well as on those elements that changed after it. To remove the `junos:changed="changed"` attribute from elements that changed before the commit, you must eliminate the cause of the warning, and commit the configuration again.

commit-scripts

Request that the NETCONF or Junos XML protocol server display commit-script-style XML data. The value of the attribute determines the output.

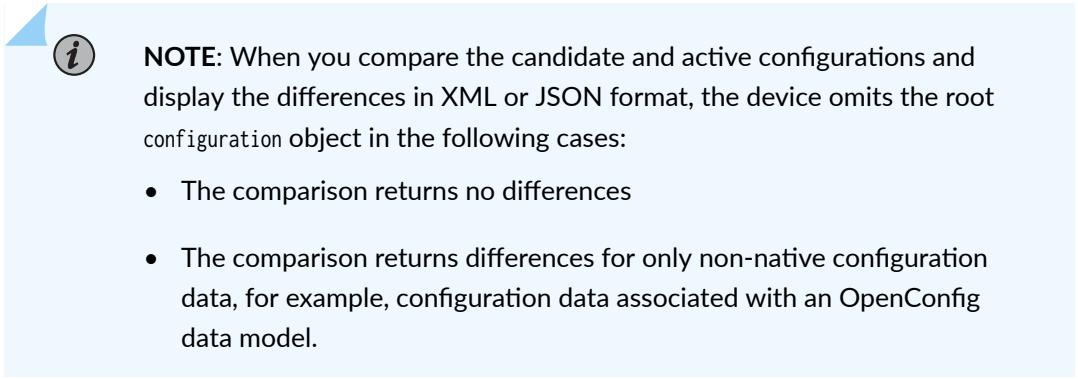
- **Values:**
  - `apply`—Display the configuration with commit script changes applied, including both transient and non-transient changes. The output is equivalent to the `| display commit-scripts` output in the CLI.
  - `apply-no-transients`—Display the configuration with commit script changes applied, but exclude transient changes. The output is equivalent to the `| display commit-scripts no-transients` output in the CLI.
  - `view`—Display the configuration in the XML format that is input to a commit script. The output is equivalent to viewing the configuration with the following attributes applied: `inherit="inherit"`, `groups="groups"`, and `changed="changed"`. The output is equivalent to the `| display commit-scripts view` output in the CLI.

compare Request that the NETCONF or Junos XML protocol server display the differences between the active or candidate configuration and a previously committed configuration (the comparison configuration). By default, the comparison uses the candidate configuration. Include the `database` attribute to specify the active configuration.

- **Values:**
  - `configuration-revision`—Reference the comparison configuration by its configuration revision ID string, which you define in the `configuration-revision="`*`revision-id`*`"` attribute.
  - `rollback`—Reference the comparison configuration by its rollback index, which you define in the `rollback="`*`rollback-number`*`"` attribute.

If you include the `compare` attribute but either omit the corresponding `configuration-revision` or `rollback` attribute or provide an invalid configuration revision ID, the server uses the most recently committed configuration as the comparison configuration.

When you compare the candidate configuration to the active configuration, the `compare` operation returns XML output. However, you can include the `format` attribute to display the differences in text, XML, or JSON format. For all other comparisons, the server returns the output as text using a patch format.



NOTE: When you compare the candidate and active configurations and display the differences in XML or JSON format, the device omits the root `configuration` object in the following cases:

- The comparison returns no differences
- The comparison returns differences for only non-native configuration data, for example, configuration data associated with an OpenConfig data model.

database Specify the configuration database from which to display data.

- **Default:** `candidate`
- **Values:**
  - `candidate`—The candidate configuration
  - `committed`—The active configuration (the one most recently committed)

If you include both the `database` and the `database-path` attributes, the `database` attribute takes precedence.

database-path

Within a commit script, this attribute specifies the path to the session's pre-inheritance candidate configuration. 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.

- **Values:** `$junos-context/commit-context/database-path`

If you include both the `database` and the `database-path` attributes, the `database` attribute takes precedence.

format

Specify the format in which the NETCONF or Junos XML protocol server returns the configuration data.

- **Default:** `xml`
- **Values:**
  - `json`—Configuration data format is JSON.

    NOTE: Integers in Junos OS configuration data emitted in JSON format are not enclosed in quotation marks.

  - `set`—Configuration data format is Junos OS configuration mode commands.
  - `text`—Configuration data format is ASCII text, which uses the newline character, tabs and other white space, braces, and square brackets to indicate the hierarchical relationships between the statements.
  - `xml`—Configuration data format is Junos XML.

NOTE: Starting in Junos OS Release 21.1R1 and Junos OS Evolved Release 22.3R1, NETCONF sessions additionally support the `json-minified` and `xml-minified` formats, which return the respective format with unnecessary spaces, tabs, and newlines removed.

groups

Request that the `junos:group="`*`group-name`*`"` attribute appear 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 from which that element was inherited.

- **Values:** `groups`

When you specify the `groups` attribute, you must also specify the `inherit` attribute.

inherit Specify how the NETCONF or Junos XML protocol server displays statements that are defined in configuration groups and interface ranges. If you omit the `inherit` attribute, the output uses the `<groups>`, `<apply-groups>`, and `<apply-groups-except>` tags to represent user-defined configuration groups and uses the `<interface-range>` tag to represent user-defined interface ranges. The output does not include tag elements for statements defined in the junos-defaults group.

- **Values:**
  - `defaults`—The output does not include the `<groups>`, `<apply-groups>`, and `<apply-groups-except>` tags, but instead displays 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>` tags, but instead displays 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 Request that the `junos:interface-ranges="`*source-interface-range*`"` attribute appear 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.

- **Values:** `interface-ranges`

When you specify the `interface-ranges` attribute, you must also specify the `inherit` attribute.

junos:key | key Request that the `junos:key="key"` attribute appear in the opening tag of each element that serves as an identifier for a configuration object.

- **Values:** `key`

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

`interface-ranges` attribute added in Junos OS Release 10.3R1.

`commit-scripts` attribute values `apply` and `apply-no-transients` added in Junos OS Release 12.1

`database-path` attribute added in Junos OS Release 12.2.

`format` attribute value `json` added in Junos OS Release 14.2.

`format` attribute value `set` added in Junos OS Release 15.1.

Starting in Junos OS Release 16.1, devices running Junos OS emit JSON-formatted configuration data using a new default implementation for serialization.

Starting in Junos OS Releases 16.1R4, 16.2R2, and 17.1R1, integers in Junos OS configuration data emitted in JSON format are not enclosed in quotation marks.

`compare` attribute value `configuration-revision` added in Junos OS Release 20.4R1 and Junos OS Evolved Release 20.4R1.

`format` attribute values `json-minified` and `xml-minified` added for NETCONF sessions only in Junos OS Release 21.1R1 and Junos OS Evolved Release 22.3R1.

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

junos:changed | 180

junos:group | 186

junos:interface-range | 187

junos:key | 188

<kill-session>

IN THIS SECTION

Usage

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

```
    </kill-session>
</rpc>
```

Description

Request that the Junos XML protocol server terminate another CLI session or Junos XML protocol session. This operation might be used when a user or application for another session holds a lock on the candidate configuration, preventing the current client application from locking the configuration.

The client application must have the Junos OS `maintenance` permission to perform this operation.

Contents

<session-id> PID of the entity conducting the session to terminate. The PID is reported in the `<xnm:error>` element that the Junos XML protocol server generates when it cannot lock a configuration as requested.

RELATED DOCUMENTATION

Terminate a Junos XML Protocol Session | 98

<xnm:error> | 166

<load-configuration>

IN THIS SECTION

Usage

```
<rpc>
    <load-configuration configuration-revision="revision-id"/>

    <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="(merge | override | update)"]
            format="json" />

    <load-configuration url="url" action="set" format="text"/>

    <load-configuration [action="(merge | override | replace | update)"]
            [format="xml"]  >
        <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="(merge | override | update)"] format="json">
        <configuration-json>
            <!-- JSON configuration data to load -->
        </configuration-json>
    </load-configuration>

    <load-configuration action="set" format="text" >
        <configuration-set>
            <!-- configuration mode commands to load -->
        </configuration-set>
```

```
    </load-configuration>
</rpc>
```

Description

Request that the NETCONF or Junos XML protocol server load configuration data into the candidate configuration or open configuration database.

If a client application issues the Junos XML protocol <open-configuration> operation to open a specific configuration database before executing the <load-configuration> operation, the server loads the configuration data into the open configuration database. Otherwise, the server loads the configuration data into the candidate configuration.

Table 9 on page 132 describes the common load configuration operations.

Table 9: Load Configuration Data

<load-configuration> Operation	**Description**
<load-configuration configuration-revision="*revision-id*"/>	Load a previously committed configuration by referencing its configuration revision ID. The specified configuration completely replaces the candidate configuration.
<load-configuration rescue="rescue"/>	Load the rescue configuration. The rescue configuration completely replaces the candidate configuration.
<load-configuration rollback="*index*"/>	Load a previously committed configuration by referencing its numerical rollback index. The specified configuration completely replaces the candidate configuration.
<load-configuration url="*url*" format="(text | xml | json)/> <load-configuration url="*url*" action="set" format="text"/>	Load configuration data from the file specified in the url attribute. Specify the full path of the file that contains the configuration data to load and the format of the data in the file. For example: <load-configuration url="/tmp/add.conf" format="text"/>

Table 9: Load Configuration Data *(Continued)*

<load-configuration> Operation	Description
<load-configuration format="xml"> <configuration>...</configuration> </load-configuration> <load-configuration format="text"> <configuration-text>...</configuration-text> </load-configuration> <load-configuration format="json"> <configuration-json>...</configuration-json> </load-configuration> <load-configuration action="set" format="text"> <configuration-set>...</configuration-set> </load-configuration>	Load the configuration as a data stream. Enclose the configuration data in the appropriate set of tags for the format.

Attributes

action

Specify how to load the configuration data, particularly when the target configuration database and the loaded configuration contain conflicting statements.

The ephemeral configuration database supports all of the `action` attribute values. The `update` value is supported in Junos OS Release 21.1R1 and later.

- **Default:** `merge`
- **Values:**
  - `merge`—Combine the data in the loaded configuration with the data in the target configuration. If statements in the loaded configuration conflict with statements in the target configuration, the loaded statements replace those statements in the target configuration.
  - `override`—Discard the entire candidate configuration and replace it with the loaded configuration. When you commit the configuration, all system processes parse the new configuration.

- `replace`—Substitute each hierarchy level or configuration object defined in the loaded configuration for the corresponding level or object in the candidate configuration.

  If the configuration data format is ASCII text, place the `replace:` statement on the line directly preceding the statements that represent the hierarchy level or object to replace. If the configuration data is Junos XML elements, include the `replace="replace"` attribute in the opening tags of the elements that represent the hierarchy levels or objects to replace.

- `set`—Load configuration data formatted as Junos OS configuration mode commands. This option executes the configuration instructions line by line. You can store the instructions in a file named by the `url` attribute, or you can enclose the instructions in a `<configuration-set>` element to provide a data stream. The instructions can contain any configuration mode command, such as `set`, `delete`, `edit`, or `deactivate`. When using the `set` action, the default and only acceptable value for the `format` attribute is "text".

- `update`—Compare a complete loaded configuration against the 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 you commit the configuration, only system processes that are affected by the changed configuration elements parse the new configuration.

configuration-revision Load a previously committed configuration by referencing its configuration revision ID. The specified configuration completely replaces the candidate configuration.

format Specify the format used for the configuration data.

- **Default:**
  - `xml`, for all `action` values except `set`
  - `text`, when `action="set"`
- **Values:**
  - `json`–Indicate that the configuration data format is JSON.
  - `text`—Indicate that the configuration data format is ASCII text or 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. Junos devices use this format for configuration files stored on the device and for the output of the CLI `show configuration` command.

The `set` command format consists of Junos OS configuration mode commands. You can view this format using the `show configuration | display set` CLI command. To load configuration mode commands, you must set the `action` attribute to "set".

- `xml`—Indicate that the configuration data is Junos XML elements.

rescue Replace the candidate configuration with the rescue configuration.

- **Values:** `rescue`

> **NOTE**: You can also use the <rollback-config> RPC to load a previously committed configuration. The <rollback-config> RPC is useful for applications that do not support executing RPCs that include XML attributes.

rollback Load a previously committed configuration by referencing its numerical rollback index. Valid values are 0 (for the most recently committed configuration) through one less than the number of stored previous configurations (maximum is 49).

> **NOTE**: You can also use the <rollback-config> RPC to load a previously committed configuration. The <rollback-config> RPC is useful for applications that do not support executing RPCs that include XML attributes.

url Specify 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 an HTTP URL.

- **Syntax:**
  - Local filename:
    - */path/filename*—File on a mounted file system, either on the local flash drive 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.

- **ftp://*username*:*password*@*hostname*/*path*/*filename***

- **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*/%2F*path*/*filename***.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

`action` attribute value `set` added in Junos OS Release 11.4.

`format` attribute value `json` added in Junos OS Release 16.1.

`configuration-revision` attribute added in Junos OS Release 20.4R1 and Junos OS Evolved Release 20.4R1.

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

<load-configuration-results> | 161

replace | 196

<lock-configuration/>

IN THIS SECTION

Usage

```
<rpc>
    <lock-configuration/>
</rpc>
```

Description

Request that the NETCONF or Junos XML protocol server open and lock the candidate configuration. This operation enables the client application to read and change the candidate configuration while preventing other users or applications from changing it. The application must emit the <unlock-configuration/> tag to unlock the configuration.

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.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94

<unlock-configuration/> | 143

<open-configuration>

IN THIS SECTION

Usage

```
<rpc>
    <open-configuration>
        <private/>
    </open-configuration>

    <open-configuration>
        <ephemeral/>
    </open-configuration>

    <open-configuration>
        <ephemeral-instance>instance-name</ephemeral-instance>
    </open-configuration>
</rpc>
```

Description

Create a private copy of the candidate configuration database or open the default instance or a user-defined instance of the ephemeral configuration database.

NOTE: Before opening a user-defined instance of the ephemeral configuration database, you must first enable the instance by configuring the `instance` *instance-name* statement at the `[edit system configuration-database ephemeral]` hierarchy level on the device.

A client application can perform the same operations on the private copy or ephemeral instance as on the regular candidate configuration, including load and commit operations. There are, however, restrictions on these operations. For details, see "<load-configuration>" on page 130 and "<commit-configuration>" on page 115.

To close a private copy or ephemeral instance and discard all uncommitted changes, execute the `<close-configuration/>` operation. Changes to the private copy or ephemeral instance 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 the changes other than by performing a commit operation, for example, by emitting the `<commit-configuration/>` tag.

NOTE: The Junos XML protocol <open-configuration> operation does not emit an "uncommitted changes will be discarded on exit" warning message when opening a private copy of the candidate configuration. However, the device still discards the uncommitted changes upon closing the private copy.

Contents

<private/>	Open a private copy of the candidate configuration database.
<ephemeral/>	Open the default instance of the ephemeral configuration database.
<ephemeral-instance>	Open the specified instance of the ephemeral configuration database. This instance must already be configured at the [edit system configuration-database ephemeral] hierarchy level on the device.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

<ephemeral> and <ephemeral-instance> elements added in Junos OS Release 16.2R2.

RELATED DOCUMENTATION

Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94

<request-end-session/>

IN THIS SECTION

Usage

```
<rpc>
    <request-end-session/>
</rpc>
```

Description

Request that the NETCONF or Junos XML protocol server end the current session.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

<end-session/> | 160

<request-login>

IN THIS SECTION

Usage

```
<rpc>
    <request-login>
        <username>account</username>
        <challenge-response>password</challenge-response>
    </request-login>
</rpc>
```

Description

Request authentication by the Junos XML protocol server when using the cleartext or SSL access protocol.

Emitting both the <username> and <challenge-response> elements is appropriate if the client application:

- Automates access to device information and does not interact with users
- Obtains the password from a user before beginning the authentication process.

Emitting only the <username> tag element is appropriate if the application does not obtain the password until the authentication process has already begun. In this case, the Junos XML protocol server returns the <challenge> tag to request the password associated with the account.

NOTE: You must escape any XML special characters in the username or password elements of a <request-login> RPC request. The following five symbols are considered special characters: greater than (>), less than (<), single quote ('), double quote ("), and ampersand (&). Both entity references and character references are acceptable escape sequence formats. For example, & and & are valid representations of an ampersand.

Contents

<challenge-response> Specify the password for the account named in the <username> element. Omit this element to instruct the Junos XML protocol server to emit the <challenge> tag to request the password.

<username> Name of the user account under which to authenticate with the Junos XML protocol server. The account must already be configured on the device where the Junos XML protocol server is running.

Release Information

XML special characters in the username or password elements must be escaped starting in Junos OS Releases 13.3R7, 14.1R6, 14.2R4, 15.1R2, and 16.1R1.

RELATED DOCUMENTATION

- Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections | 80
- <challenge> | 151
- <rpc> | 142

<rpc>

IN THIS SECTION

Usage

```
<junoscript>
    <rpc [attributes]>
        <!-- tag elements in a request from a client application -->
    </rpc>
</junoscript>
```

Description

Enclose all tag elements in a request generated by a client application.

Attributes

(Optional) One or more attributes of the form *attribute-name*="*value*".

A client application can use <rpc> attributes to associate requests and responses by assigning a unique attribute value in the opening <rpc> tag. The Junos XML protocol server echoes the attribute unchanged in its opening <rpc-reply> tag, making it simple to map the response to the initiating request.

NOTE: The xmlns:junos attribute name is reserved. The Junos XML protocol server sets the attribute to an appropriate value on the opening <rpc-reply> tag, so client applications must not emit it on the opening <rpc> tag.

RELATED DOCUMENTATION

Send Requests to the Junos XML Protocol Server | 84

<junoscript> | 146

<rpc-reply> | 165

<unlock-configuration/>

IN THIS SECTION

Usage

```
<rpc>
    <unlock-configuration/>
</rpc>
```

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.

Release Information

This operation 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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol | 94

<lock-configuration/> | 136

CHAPTER 8

Junos XML Protocol Processing Instructions

IN THIS CHAPTER

<?xml?>

IN THIS SECTION

Usage

```
<?xml version="version" encoding="encoding"?>
```

Description

Specify the XML version and character encoding scheme for the session.

Attributes

encoding Specify the standardized character set that the emitter uses and can interpret.

version Specify the version of XML used by the emitter.

RELATED DOCUMENTATION

Start a Junos XML Protocol Session | 75

<junoscript> | 146

<junoscript>

IN THIS SECTION

Usage

```
<!-- emitted by a client application -->
<junoscript version="version" [hostname="hostname"] [junos:key="key"]
        [release="release"]>
    <!-- all tag elements generated by the application during the session -->
</junoscript>

<!-- emitted by the Junos XML protocol server -->
<junoscript xmlns="namespace-URL" xmlns:junos="namespace-URL"
        schemaLocation="namespace-URL" os="os" release="release"
        hostname="hostname" version="version">
    <!-- all tag elements generated by the Junos XML protocol server during the session -->
</junoscript>
```

Description

Enclose all tag elements in a Junos XML protocol session (act as the root tag element for the session). The client application and Junos XML protocol server each emit this tag element, enclosing all other tag elements that they emit during a session inside it.

Attributes

hostname Name of the device on which the tag element's originator is running.

junos:key Request that the Junos XML protocol server include the `junos:key="key"` attribute in the opening tag of each tag element that serves as an identifier for a configuration object.

os Operating system of the device named by the `hostname` attribute.

release Identify the Junos OS release being run by the tag element's originator. Software modules always set this attribute, but client applications are not required to set it.

schemaLocation XML namespace for the XML Schema-language representation of the Junos OS configuration hierarchy.

version (Required for a client application) Specify the version of the Junos XML management protocol used for the enclosed set of tag elements.

xmlns XML namespace for the tag elements enclosed by the `<junoscript>` tag element that do not have a prefix on their names (that is, the default namespace for Junos XML tag elements). The value is a URL of the form `http://xml.juniper.net/xnm/`*version-code*`/xnm`, where *version-code* is a string such as 1.1.

xmlns:junos XML namespace for the tag elements enclosed by the `<junoscript>` tag element that have the `junos:` prefix. The value is a URL of the form `http://xml.juniper.net/junos/`*release-code*`/junos`, where *release-code* is the standard string that represents a release of the Junos OS, such as 20.4R1 for the initial version of Junos OS Release 20.4.

RELATED DOCUMENTATION

Start a Junos XML Protocol Session | **75**

Request Identifiers for Configuration Elements Using the Junos XML Protocol | **406**

<rpc> | **142**

<rpc-reply> | **165**

junos:key | **188**

CHAPTER 9

Junos XML Protocol Response Tags

IN THIS CHAPTER

<abort-acknowledgement/>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <any-child-of-rpc-reply>
        <abort-acknowledgement/>
    </any-child-of-rpc-reply>
</rpc-reply>
```

Description

Indicates 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.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

<abort/> | 113

<authentication-response>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <authentication-response>
        <status>authentication-outcome</status>
        <message>message</message>
        <login-name>remote-username</login-name>
    </authentication-response>
</rpc-reply>
```

Description

Indicate whether an authentication attempt succeeded. The Junos XML protocol server returns the tag element in response to the <request-login> tag element emitted by a client application that uses the clear-text or Secure Sockets Layer (SSL) access protocol.

Contents

<login-name> Specifies the username that the client application provided to an authentication utility such as RADIUS or TACACS+. This tag element appears only if the username that it contains differs from the username contained in the <message> tag element.

<message> Names the account under which a connection to the Junos XML protocol server is established, if authentication succeeds. If authentication fails, explains the reason for the failure.

<status> Indicates whether the authentication attempt succeeded. There are two possible values:

- fail—The attempt failed. The Junos XML protocol server also emits the <challenge> tag element to request the password again, up to a maximum of three attempts.
- success—The attempt succeeded. An authenticated connection to the Junos XML protocol server is established.

RELATED DOCUMENTATION

Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections | 80

<challenge> | 151

<request-login> | 140

<rpc-reply> | 165

<challenge>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <challenge echo="no">Password:</challenge>
</rpc-reply>
```

Description

Request the password associated with an account during authentication with a client application that uses the clear-text or SSL access protocol. The Junos XML protocol server emits this tag element when the initial <request-login> tag element emitted by the client application does not enclose a <challenge-response> tag element, and when the password enclosed in a <challenge-response> tag element is incorrect (in the latter case, the server also emits an <authentication-response> tag element enclosing child tag elements that indicate the password is incorrect).

The tag element encloses the string Password: which the client application can forward to the screen as a prompt for a user.

Attributes

echo Specifies whether the password string typed by the user appears on the screen. The value "no" specifies that it does not.

RELATED DOCUMENTATION

Authenticate with the Junos XML Protocol Server for Cleartext or SSL Connections | 80

Satisfy the Prerequisites for Establishing a Connection to the Junos XML Protocol Server | 60

<authentication-response> | 149

<request-login> | 140

<rpc-reply> | 165

<checksum-information>

IN THIS SECTION

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>
```

Description

Encloses 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.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

<get-checksum-information> | 122

<commit-results>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <!-- for the candidate configuration or ephemeral 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>
```

Description

Tag element returned by the Junos XML protocol server in response to a <commit-configuration> request by a client application. The <commit-results> element contains information about the requested commit operation performed by the server on a particular Routing Engine.

Contents

<load-success/> 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.

The <routing-engine> tag element is described separately.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | 288

<commit-configuration> | 115

<routing-engine> | 163

<commit-revision-information>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>

            <!-- configuration with commit revision identifier -->
            <commit-revision-information>
                <old-db-revision>old-revision-id</old-db-revision>
                <new-db-revision>new-revision-id</new-db-revision>
            </commit-revision-information>

        </routing-engine>
    </commit-results>

</rpc-reply>
```

Description

Child element included in a Junos XML protocol server <commit-results> response element to return information about the old and new configuration revision identifiers (CRI) on a particular Routing Engine. The CRI is a unique string (for example, re0-1365168149-1) that is associated with a committed configuration.

Network management system (NMS) applications, such as Junos Space, can use the configuration revision identifier to determine if the NMS's known configuration for a Junos device is identical to the device's current configuration. The NMS can detect if out-of-band commits were made to the device by comparing the CRI associated with the NMS's last commit to the CRI of the configuration on the device.

Contents

<old-db-revision> Indicates the old configuration revision identifier, which is the identifier of the configuration prior to the previously successfully committed configuration.

<new-db-revision> Indicates the new configuration revision identifier, which is the identifier of the last successfully committed configuration.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

Element introduced in Junos OS Release 16.1.

RELATED DOCUMENTATION

View the Configuration Revision Identifier for Determining Synchronization Status of Devices with NMS | **307**

<commit-results> | **153**

<routing-engine> | **163**

<database-status>

IN THIS SECTION

Usage

```
<junoscript>
    <any-child-of-junoscript>
        <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>
    </any-child-of-junoscript>
</junoscript>
```

Description

Describe a user or Junos XML protocol 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

<commit-at/> Indicates that the user has scheduled a commit operation for a later time.

<edit-path> Specifies the user's current location in the configuration hierarchy, in the form of the CLI configuration mode banner.

<exclusive/> 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 `<lock-configuration/>` tag element.

<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

Handle an Error or Warning in Junos XML Protocol Sessions | 92

<database-status-information> | 159

<junoscript> | 146

<xnm:error> | 166

<database-status-information>

IN THIS SECTION

Usage

```
<junoscript>
    <any-child-of-junoscript>
        <xnm:error>
            <database-status-information>
                <database-status>...</database-status>
            </database-status-information>
        </xnm:error>
    </any-child-of-junoscript>
<junoscript>
```

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

- Handle an Error or Warning in Junos XML Protocol Sessions | 92
- <database-status> | 157
- <junoscript> | 146
- <xnm:error> | 166

<end-session/>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <end-session/>
</rpc-reply>
```

Description

Indicates 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.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

End a Junos XML Protocol Session and Close the Connection | 100

<request-end-session/> | 139

<load-configuration-results>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <load-configuration-results>
        <load-success/>
        <load-error-count>errors</load-error-count>
    </load-configuration-results>
</rpc-reply>
```

Description

Tag element returned by the NETCONF or Junos XML protocol server in response to a <load-configuration> request by a client application.

In a Junos XML protocol session, the <load-configuration-results> element encloses either a <load-success/> tag or a <load-error-count> tag, which indicates the success or failure of the load configuration operation. In a NETCONF session, the <load-configuration-results> element encloses either an <ok/> tag or a <load-error-count> tag to indicate the success or failure of the load configuration operation.

Contents

<load-error-count>	Specifies the number of errors that occurred when the server attempted to load new data into the candidate configuration or open configuration database. The target configuration must be restored to a valid state before it is committed.
<load-success/>	Indicates that the server successfully loaded new data into the candidate configuration or open configuration database.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

<load-configuration> | 130

<reason>

IN THIS SECTION

Usage

```
<xnm:error | xnm:warning>
    <reason>
         <daemon>process</daemon>
        <process-not-configured/>
        <process-disabled/>
        <process-not-running/>
    </reason>
</xnm:error | xnm:warning>
```

Description

Child element included in an <xnm:error> or <xnm:warning> element in a Junos XML 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

- Handle an Error or Warning in Junos XML Protocol Sessions | 92
- <xnm:error> | 166
- <xnm:warning> | 169

<routing-engine>

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <commit-results>

    <!-- when the candidate configuration or private copy is committed -->
        <routing-engine>
            <name>reX</name>
            <commit-success/>
```

```
            <commit-revision-information>
                <old-db-revision>old-revision-id</old-db-revision>
                <new-db-revision>new-revision-id</new-db-revision>
            </commit-revision-information>
        </routing-engine>

    <!-- when the candidate configuration or private copy is syntactically valid -->
        <routing-engine>
            <name>reX</name>
            <commit-check-success/>
        </routing-engine>

    <!-- when an instance of the ephemeral database is committed -->
        <routing-engine>
            <name>reX</name>
            <commit-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

Description

Child element included in a Junos XML protocol server <commit-results> response element to return information about a requested commit operation on a particular Routing Engine.

Contents

<commit-check-success>	Indicates that the configuration is syntactically correct.
<commit-success>	Indicates that the Junos XML protocol server successfully committed the configuration.
<name>	Name of the Routing Engine on which the commit operation was performed. Possible values are re0 and re1.

The <commit-revision-information> tag element is described separately.

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 on Junos devices that identify the URI http://xml.juniper.net/netconf/junos/1.0 in the capabilities exchange.

RELATED DOCUMENTATION

<commit-results> | 153

<commit-revision-information> | 155

<rpc-reply>

IN THIS SECTION

Usage

```
<junoscript>
    <rpc-reply xmlns:junos="namespace-URL">
        <!-- tag elements in a reply from the Junos XML protocol server -->
    </rpc-reply>
</junoscript>
```

Description

Enclose all tag elements in a reply from the Junos XML protocol server. The immediate child tag element is usually one of the following:

- The tag element used to enclose data generated by the Junos XML protocol server or a Junos OS module in response to a client application's request.

- The <output> tag element, if the Junos XML API does not define a specific tag element for the requested information.

Attributes

xmlns:junos Names the XML namespace for the Junos XML tag elements enclosed by the <rpc-reply> tag element that have the junos: prefix. The value is a URL of the form http://xml.juniper.net/junos/*release-code*/junos, where *release-code* is the standard string that represents a Junos OS release, such as 20.4R1 for the initial version of Junos OS Release 20.4.

RELATED DOCUMENTATION

Parse the Junos XML Protocol Server Response | 87

<junoscript> | 146

<rpc> | 142

<xnm:error>

IN THIS SECTION

Usage

```
<junoscript>
    <any-child-of-junoscript>
        <xnm:error xmlns="namespace-URL" xmlns:xnm="namespace-URL">
            <parse/>
            <source-daemon>module-name </source-daemon>
            <filename>filename</filename>
```

```
            <line-number>line-number </line-number>
            <column>column-number</column>
            <token>input-token-id </token>
            <edit-path>edit-path</edit-path>
            <statement>statement-name </statement>
            <message>error-string</message>
            <re-name>re-name-string</re-name>
            <database-status-information>...</database-status-information>
            <reason>...</reason>
        </xnm:error>
    </any-child-of-junoscript>
</junoscript>
```

Description

Indicate that the Junos XML protocol 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 Junos XML protocol server does not necessarily emit all child tag elements; it omits tag elements that are not relevant to the current request.

Attributes

xmlns 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 XML namespace for child tag elements that have the `xnm:` prefix on their names. The value is a URL of the form `http://xml.juniper.net/xnm/`*version*`/xnm`, where *version* is a string such as "1.1".

Contents

<column> (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.

<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.
<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 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

- Handle an Error or Warning in Junos XML Protocol Sessions | **92**
- <database-status-information> | **159**
- <junoscript> | **146**
- <reason> | **162**
- <xnm:warning> | **169**

<xnm:warning>

IN THIS SECTION

Usage

```
<junoscript>
    <any-child-of-junoscript>
        <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>
    </any-child-of-junoscript>
</junoscript>
```

Description

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.

Attributes

xmlns 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 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

Handle an Error or Warning in Junos XML Protocol Sessions | 92

CHAPTER 10

Junos XML Element Attributes

IN THIS CHAPTER

active

IN THIS SECTION

Usage

```
<rpc>
   <load-configuration>
       <configuration>
           <!-- opening tag for each parent of the element -->
               <element active="active">
                   <name>identifier</name> <!-- if element has identifier -->
               </element>
           <!-- closing tag for each parent of the element -->
       </configuration>
   </load-configuration>
</rpc>
```

Description

Reactivate a previously deactivated configuration element.

The `active` attribute can be combined with one or more of the `insert`, `rename`, or `replace` attributes. To deactivate an element, include the `inactive` attribute instead.

RELATED DOCUMENTATION

- Change a Configuration Element's Activation State Using the Junos XML Protocol | 268
- Change a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol | 275
- inactive | 177
- insert | 178

<load-configuration> | 130

rename | 195

replace | 196

<rpc> | 142

count

IN THIS SECTION

Usage

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object -->
                <object-type count="count"/>
            <!-- closing tags for each parent of the object -->
        </configuration>
    </get-configuration>
</rpc>
```

Description

Specify the number of configuration objects of the specified type about which to return information. If the attribute is omitted, the Junos XML protocol server returns information about all objects of the type.

The attribute can be combined with one or more of the `matching`, `recurse`, and `start` attributes.

If the application requests Junos XML-tagged output (the default), the Junos XML protocol server includes two attributes for each returned object:

- `junos:position`—Specifies the object's numerical index.

- junos:total—Reports the total number of such objects that exist in the hierarchy.

These attributes do not appear if the application requests formatted ASCII output by including the format="text" attribute in the opening <get-configuration> tag.

RELATED DOCUMENTATION

Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420
<get-configuration> | 123
matching | 191
recurse | 194
<rpc> | 142
start | 200

delete

IN THIS SECTION

Usage

```
<rpc>
    <load-configuration>
        <configuration>
            <!-- opening tag for each parent of the element -->

            <!-- For a hierarchy level or object without an identifier -->
                <level-or-object delete="delete">

            <!-- For an object with an identifier (here, called <name>) -->
                <object delete="delete">
                    <name>identifier</name>
```

```
            </object>

        <!-- For a single-value or fixed-form option of an object -->
            <object>
                <name>identifier</name> <!-- if object has identifier -->
                <option delete="delete"/>
            </object>

        <!-- closing tag for each parent of the element -->

        <!-- For a value in a multivalued option of an object -->
        <!-- opening tag for each parent of the parent object -->
            <parent-object>
                <name>identifier</name>
                <object delete="delete">value</object>
            </parent-object>
    <!-- closing tag for each parent of the parent object -->

    </configuration>
</load-configuration>
</rpc>
```

Description

Specify that the Junos XML protocol server remove the configuration element from the candidate configuration or open configuration database. The only acceptable value for the attribute is "delete".

RELATED DOCUMENTATION

Delete Elements in Configuration Data Using the Junos XML Protocol | 244

<load-configuration> | 130

<rpc> | 142

inactive

IN THIS SECTION

Usage

```
<rpc>
   <load-configuration>
       <configuration>
           <!-- opening tag for each parent of the element -->

              <!-- if immediately deactivating a newly created element -->
               <element inactive="inactive">
                   <name>identifier</name> <!-- if element has identifier -->
                   <!-- tag elements for each child of the element -->
               </element>

              <!-- if deactivating an existing element -->
               <element inactive="inactive">
                   <name>identifier</name> <!-- if element has identifier -->
               </element>

           <!-- closing tag for each parent of the element -->
       </configuration>
   </load-configuration>
</rpc>
```

Description

Deactivate a configuration element when loading configuration data into the candidate configuration or open configuration database using the <load-configuration> operation. When the configuration is committed, the element remains in the configuration, but the element does not affect the functioning of the routing, switching, or security platform.

The inactive attribute can be combined with one or more of the insert, rename, or replace attributes, as described in "Changing a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol" on page 275. To reactivate a deactivated element, include the active attribute instead.

RELATED DOCUMENTATION

- Change a Configuration Element's Activation State Using the Junos XML Protocol | 268
- active | 173
- insert | 178
- <load-configuration> | 130
- rename | 195
- <rpc> | 142

insert

IN THIS SECTION

Usage

```
<rpc>
   <load-configuration>
      <configuration>
         <!-- opening tag for each parent of the set -->

            <ordered-set insert="first">
                <name>identifier-for-moving-object</name>
            </ordered-set>

            <!-- if each element in the ordered set has one identifier -->
            <ordered-set insert="(before | after)" name="referent-value">
```

```
            <name>identifier-for-moving-object</name>
        </ordered-set>

        <!-- if each element in the ordered set has two identifiers -->
        <ordered-set insert="(before | after)"
         identifier1="referent-value" identifier2="referent-value">
            <identifier1>value-for-moving-object</identifier1>
            <identifier2>value-for-moving-object</identifier2>
        </ordered-set>

      <!-- closing tag for each parent of the set -->
    </configuration>
  </load-configuration>
</rpc>
```

Description

Change the position of an existing configuration element in an ordered set. The `insert="first"` attribute moves the existing element to the first position in the list. The `insert="before"` and `insert="after"` attributes specify the new position relative to a reference element, which is specified by including an attribute named after each of its identifier tags. In the Usage section, the identifier tag element is called `<name>` when each element in the set has one identifier.

The `insert` attribute can be combined with either the `active` or `inactive` attribute, as described in "Changing a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol" on page 275.

RELATED DOCUMENTATION

Reorder Elements In Configuration Data Using the Junos XML Protocol | 258

active | 173

inactive | 177

<load-configuration> | 130

<rpc> | 142

junos:changed

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration standard-attributes junos:changed="changed">
        <!-- opening-tag-for-each-parent-level junos:changed="changed" -->

            <!-- If the changed element is an empty tag -->
            <element junos:changed="changed"/>

            <!-- If the changed element has child tag elements -->
            <element junos:changed="changed">
                <first-child-of-element junos:changed="changed">
                <second-child-of-element junos:changed="changed">
                <!-- additional children of element -  ->
            </element>

        <!-- closing-tag-for-each-parent-level -->
    </configuration>
</rpc-reply>
```

Description

Indicate that a configuration element has changed since the last commit operation. The Junos XML protocol server includes the attribute when the client application includes the `changed` attribute in a `<get-configuration>` operation. The attribute appears in the opening tag of every parent tag element in the path to the changed configuration element, including the opening top-level `<configuration>` tag.

The attribute does not appear if the client requests formatted ASCII output by including the `format="text"` attribute in the empty `<get-configuration/>` tag or opening `<get-configuration>` tag.

For information about the standard attributes in the opening <configuration> tag, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

RELATED DOCUMENTATION

Request Change Indicators for Configuration Elements Using the Junos XML Protocol | 411

<get-configuration> | 123

<rpc-reply> | 165

junos:changed-localtime

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration 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>
```

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.

RELATED DOCUMENTATION

Specify the Database Source for Configuration Data to Return | 377

<rpc-reply> | 165

junos:changed-seconds | 182

junos:changed-seconds

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration 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>
```

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.

RELATED DOCUMENTATION

Specify the Database Source for Configuration Data to Return | 377

<rpc-reply> | 165

junos:changed-localtime | 181

junos:commit-localtime

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration 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.

RELATED DOCUMENTATION

- Specify the Database Source for Configuration Data to Return | 377
- <rpc-reply> | 165
- junos:commit-user | 185
- junos:commit-seconds | 184

junos:commit-seconds

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration 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 number of seconds since midnight on 1 January 1970.

RELATED DOCUMENTATION

- Specify the Database Source for Configuration Data to Return | 377
- <rpc-reply> | 165
- junos:commit-user | 185
- junos:commit-localtime | 183

junos:commit-user

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration 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 Junos OS username of the user who requested the commit operation.

RELATED DOCUMENTATION

- Specify the Database Source for Configuration Data to Return | 377
- <rpc-reply> | 165
- junos:commit-localtime | 183
- junos:commit-seconds | 184

junos:group

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
   <configuration>
       <!-- opening tag for each parent of the element -->
           <inherited-element junos:group="source-group">
               <inherited-child-of-inherited-element junos:group="source-group">
                  <!-- inherited-children-of-child junos:group="source-group" -->
               </inherited-child-of-inherited-element>
           </inherited-element>
       <!-- closing tag for each parent of the element  -->
   </configuration>
</rpc-reply>
```

Description

Name the configuration group from which each configuration element is inherited. The Junos XML protocol server includes the attribute when the client application includes the `inherit` and `groups` attributes in a `<get-configuration>` operation.

The attribute does not appear if the client requests formatted ASCII output by including the `format="text"` attribute in the `<get-configuration>` operation. Instead, the Junos XML protocol server provides the information in a comment directly above each inherited element.

RELATED DOCUMENTATION

Specify How to Display Inheritance for Configuration Groups and Interface Ranges Using the Junos XML Protocol | 391

<get-configuration> | 123

junos:interface-range

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <interfaces>
            <!-- For each inherited element -->
            <interface junos:interface-range="source-interface-range">
                <inherited-element junos:interface-range="source-interface-range">
                    <inherited-child-of-inherited-element
                              junos:interface-range="source-interface-range">
                       <!-- inherited-children-of-child
                                 junos:interface-range="source-interface-range" -->
                    </inherited-child-of-inherited-element>
                </inherited-element>
            </interface>
        </interfaces>
    </configuration>
</rpc-reply>
```

Description

Name the interface range from which each configuration element is inherited.

The Junos XML protocol server includes the `junos:interface-range` attribute when the client application performs a `<get-configuration>` operation and includes the `inherit` and `interface-ranges` attributes.

The server response does not include the junos:interface-range attribute if the client performs a <get-configuration> operation but requests formatted ASCII output.

RELATED DOCUMENTATION

Specify How to Display Inheritance for Configuration Groups and Interface Ranges Using the Junos XML Protocol | 391

<get-configuration> | 123

junos:key

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration>
        <!-- opening tag for each parent of the object -->
            <object>
                <name junos:key="key">identifier</name>
                <!-- additional children of object -->
            </object>
       <!-- closing tag for each parent of the object  -->
    </configuration>
</rpc-reply>
```

Description

Indicate that a child configuration tag element is the identifier for its parent tag element. The Junos XML protocol server includes the attribute when the client application requests information about an object

type (with the <get-configuration> tag element) and has included the junos:key attribute in either the <get-configuration> tag or in the opening <junoscript> tag for the current session.

The attribute does not appear if the client requests formatted ASCII output by including the format="text" attribute in the empty <get-configuration/> tag or opening <get-configuration> tag.

RELATED DOCUMENTATION

Request Identifiers for Configuration Elements Using the Junos XML Protocol | 406

<get-configuration> | 123

<junoscript> | 146

<rpc> | 142

junos:position

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration>
        <!-- opening tags for each parent of the object -->
            <object junos:position="index" junos:total="total" >
        <!-- closing tags for each parent of the object -->
    </configuration>
</rpc-reply>
```

Description

Specify the index number of the configuration object in the list of objects of a specified type about which information is being returned. The Junos XML protocol server includes the attribute when the client application requests information about an object type (with the <get-configuration> tag element) and includes the count attribute, the start attribute, or both, in the opening tag for the object type.

The attribute does not appear if the client requests formatted ASCII output by including the format="text" attribute in the opening <get-configuration> tag.

RELATED DOCUMENTATION

- Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420
- count | 174
- <get-configuration> | 123
- junos:total | 190
- <rpc> | 142
- start | 200

junos:total

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <configuration>
        <!-- opening tags for each parent of the object -->
            <object junos:position="index" junos:total="total">
        <!-- closing tags for each parent of the object -->
```

```
    </configuration>
</rpc-reply>
```

Description

Specify the number of configuration objects of a specified type about which information is being returned. The Junos XML protocol server includes the attribute when the client application requests information about an object type (with the <get-configuration> tag element) and includes the count attribute, the start attribute, or both, in the opening tag for the object type.

The attribute does not appear if the client requests formatted ASCII output by including the format="text" attribute in the opening <get-configuration> tag.

RELATED DOCUMENTATION

- Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420
- count | 174
- <get-configuration> | 123
- junos:position | 189
- <rpc> | 142
- start | 200

matching

IN THIS SECTION

Usage

```
<rpc>
    <get-configuration>
```

```
        <configuration>
            <!-- opening tags for each parent of the level -->
                <level matching="matching-expression"/>
            <!-- closing tags for each parent of the level -->
    </configuration>
    </get-configuration>
</rpc>
```

Description

Request information about only those instances of a configuration object type at the specified level in the configuration hierarchy that have the specified set of characters in their identifier names (characters that match a regular expression). If the attribute is omitted, the Junos XML protocol server returns the complete set of child tag elements for the specified parent level.

The attribute can be combined with one or more of the `count`, `recurse`, and `start` attributes.

To represent the objects to return, the *matching-expression* value uses a slash-separated list of hierarchy level and object names similar to an XML Path Language (XPath) representation. Each level in the representation can be either a full level name or a regular expression that matches the identifier name of one or more instances of an object type:

```
object-type[name='regular-expression']"
```

The regular expression uses the notation defined in POSIX Standard 1003.2 for extended (modern) UNIX regular expressions. For details about the notation, see "Requesting Subsets of Configuration Objects Using Regular Expressions" on page 430.

RELATED DOCUMENTATION

Request Subsets of Configuration Objects Using Regular Expressions | 430

count | 174

<get-configuration> | 123

<rpc> | 142

start | 200

protect

IN THIS SECTION

Usage

```
<rpc>
   <load-configuration>
      <configuration>
          <!-- opening tag for each parent of the element -->
              <element protect="protect">
                  <name>identifier</name> <!-- if element has identifier -->
              </element>
          <!-- closing tag for each parent of the element -->
      </configuration>
   </load-configuration>
</rpc>
```

Description

Protect a configuration element from being modified or deleted. The `protect` attribute can be applied to configuration hierarchies or individual configuration statements. The `protect` attribute can be combined with the `active` and `inactive` attributes. To unprotect a protected element, include the `unprotect` attribute instead.

Release Information

Attribute introduced in Junos OS Release 11.2.

RELATED DOCUMENTATION

Protect or Unprotect a Configuration Object Using the Junos XML Protocol | 263

Example: Protecting the Junos OS Configuration from Modification or Deletion

<load-configuration> | 130

<rpc> | 142

unprotect | 201

recurse

IN THIS SECTION

Usage

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object -->
                <object-type recurse="false"/>
            <!-- closing tags for each parent of the object -->
        </configuration>
    </get-configuration>
</rpc>
```

Description

Request only the identifier tag element for each configuration object of a specified type in the configuration hierarchy. If the attribute is omitted, the Junos XML protocol server returns the complete set of child tag elements for every object. The only acceptable value for the attribute is "false".

The attribute can be combined with one or more of the `count`, `matching`, and `start` attributes.

RELATED DOCUMENTATION

Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol | 424

count | 174

<get-configuration> | 123

<rpc> | 142

start | 200

rename

IN THIS SECTION

Usage

```
<rpc>
    <load-configuration>
        <configuration>
            <!-- opening tag for each parent of the object -->

                <!-- if the object has one identifier -->
                <object rename="rename" name="new-name">
                    <name>current-name</name>
                </object>

                <!-- if the object has two identifiers, both changing -->
                <object rename="rename" identifier1="new-name" \
                        identifier2=new-name">
                    <identifier1>current-name</identifier1>
                    <identifier2>current-name</identifier2>
                </object>

            <!-- closing tag for each parent of the object -->
        </configuration>
```

```
    </load-configuration>
</rpc>
```

Description

Change the name of one or more of a configuration object's identifiers. In the Usage section, the identifier tag element is called <name> when the element has one identifier.

The rename attribute can be combined with either the inactive or active attribute.

RELATED DOCUMENTATION

Rename Objects In Configuration Data Using the Junos XML Protocol | 255

active | 173

inactive | 177

<load-configuration> | 130

<rpc> | 142

replace

IN THIS SECTION

Usage

```
<rpc>
    <load-configuration action="replace">
        <configuration>
            <!-- opening tag for each parent of the element -->
                <container-tag replace="replace">
                    <name>identifier</name>
```

```
                    <!-- tag elements for other children, if any -->
                </container-tag>
            <!-- closing tag for each parent of the element -->
        </configuration>
    </load-configuration>
</rpc>
```

Description

Specify that the configuration element completely replace the element that has the same identifier (in the Usage section, the identifier tag element is called <name>) in the candidate configuration or open configuration database. If the attribute is omitted, the Junos XML protocol server merges the element with the existing element as described in "Merging Elements in Configuration Data Using the Junos XML Protocol" on page 232. The only acceptable value for the attribute is "replace".

The client application must also include the action="replace" attribute in the opening <load-configuration> tag.

NOTE: Starting in Junos OS Release 18.1R1, the ephemeral configuration database supports loading configuration data using the <load-configuration> action attribute values of override and replace in addition to the previously supported values of merge and set.

The replace attribute can be combined with either the active or inactive attribute, as described in "Changing a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol" on page 275.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
18.1R1	Starting in Junos OS Release 18.1R1, the ephemeral configuration database supports loading configuration data using the <load-configuration> action attribute values of override and replace in addition to the previously supported values of merge and set.

RELATED DOCUMENTATION

Replace Elements in Configuration Data Using the Junos XML Protocol | 237

active | 173

inactive | 177
<load-configuration> | 130
<rpc> | 142

replace-pattern

IN THIS SECTION

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 placement of the attributes in the configuration data determines the scope of the replacement.

Attributes

replace-pattern="*pattern1*"	Text string or regular expression that defines the identifiers or values you want to match.
with="*pattern2*"	Text string or regular expression that replaces the identifiers and values located with *pattern1*.
upto="*n*"	Number of objects replaced. The value of *n* controls the total number of objects that the device replaces in the configuration (not the total number of times the pattern occurs). The device replaces objects at the same hierarchy level (siblings) first. The device considers multiple occurrences of a pattern within a given object as a single replacement. If you omit the `upto` attribute or if you set the attribute equal to zero, the device replaces all identifiers and values that match the pattern. • **Range:** 1 through 4294967295 • **Default:** 0

Release Information

Attribute introduced in Junos OS Release 15.1R1.

RELATED DOCUMENTATION

Replace Patterns in Configuration Data Using the NETCONF or Junos XML Protocol | 282

Modifying the Configuration for a Device

Modifying the Configuration for a Device

replace

start

IN THIS SECTION

Usage

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object -->
                <object-type start="index"/>
            <!-- closing tags for each parent of the object -->
        </configuration>
    </get-configuration>
</rpc>
```

Description

Specify the index number of the first object to return (1 for the first object, 2 for the second, and so on) when requesting information about a configuration object of a specified type. If the attribute is omitted, the returned set of objects starts with the first one in the configuration hierarchy.

The attribute can be combined with one or more of the `count`, `matching`, and `recurse` attributes.

If the application requests Junos XML-tagged output (the default), the Junos XML protocol server includes two attributes for each returned object:

- `junos:position`—Specifies the object's numerical index.

- junos:total—Reports the total number of such objects that exist in the hierarchy.

These attributes do not appear if the client requests formatted ASCII output by including the format="text" attribute in the opening <get-configuration> tag.

RELATED DOCUMENTATION

Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420

count | 174

<get-configuration> | 123

recurse | 194

<rpc> | 142

unprotect

IN THIS SECTION

Usage

```
<rpc>
   <load-configuration>
       <configuration>
           <!-- opening tag for each parent of the element -->
               <element unprotect="unprotect">
                   <name>identifier</name> <!-- if element has identifier -->
               </element>
           <!-- closing tag for each parent of the element -->
       </configuration>
```

```
    </load-configuration>
</rpc>
```

Description

Unprotect a previously protected configuration element. The `unprotect` attribute cannot be combined with other attributes such as `active`, `inactive`, `rename`, or `replace`. If an element is protected, a request to simultaneously unprotect and modify the element will unprotect the element, but it will also produce a warning message that the additional modification cannot be completed because the element is protected. You must unprotect the element first and then make the modification.

Release Information

Attribute introduced in Junos OS Release 11.2.

RELATED DOCUMENTATION

- Protect or Unprotect a Configuration Object Using the Junos XML Protocol | **263**
- Example: Protecting the Junos OS Configuration from Modification or Deletion
- <load-configuration> | **130**
- <rpc> | **142**
- protect | **193**

xmlns

IN THIS SECTION

Usage

```
<rpc-reply xmlns:junos="URL">
    <operational-response xmlns="URL-for-DTD">
        <!-- Junos XML tag elements for the requested information -->
    </operational-response>
</rpc-reply>
```

Description

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.

RELATED DOCUMENTATION

Request Operational Information Using the Junos XML Protocol | 359

<rpc-reply> | 165

PART 3

Manage Configurations Using the Junos XML Protocol

CHAPTER 11

Change the Configuration Using the Junos XML Protocol

IN THIS CHAPTER

Request Configuration Changes Using the Junos XML Protocol

SUMMARY

Client applications can use Junos XML protocol operations to modify the specified configuration database on a Junos device.

A client application can use the Junos XML protocol operations to modify the configuration on a device running Junos or a device running Junos OS Evolved. A client application can load the configuration data from a file or a data stream. The configuration data format can be Junos XML elements, formatted ASCII text, configuration mode commands, or JSON. The Junos XML tag elements described here correspond to configuration statements.

To change the configuration on a device running Junos OS or a device running Junos OS Evolved, the client application performs the procedures described in the indicated sections:

1. Establishes a connection to the Junos XML protocol server on the network device.

   See "Connect to the Junos XML Protocol Server" on page 73.

2. Starts a Junos XML protocol session, as described in "Start a Junos XML Protocol Session" on page 75.

3. Opens the target configuration database using one of the following options:

   - Opens and optionally locks the candidate configuration. Locking the candidate configuration prevents other users or applications from changing it at the same time.
   - Creates a private copy of the candidate configuration. A private copy enables the application to make changes without affecting the candidate configuration until the copy is committed.
   - Opens an instance of the ephemeral configuration database.

   For more information, see "Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol" on page 94 and "Understanding the Ephemeral Configuration Database" on page 310.

4. Loads the configuration data by enclosing the `<load-configuration>` operation in an `<rpc>` element.

   The `<load-configuration>` operation offers functionality that is analogous to configuration mode commands in the Junos OS CLI. A client application can include various attributes in the tag to specify the source and format of the configuration data. An application can completely replace the

existing candidate configuration, or it can specify how the server loads the provided data into the target configuration database. The basic syntax is as follows:

```
<rpc>
    <!-- If providing configuration data in a file -->
      <load-configuration url="file" [optional attributes] />

    <!-- If providing configuration data in a data stream -->
      <load-configuration [optional attributes] >
        <(configuration | configuration-text | configuration-set | configuration-json)>
            <!-- configuration data -->
        </(configuration | configuration-text | configuration-set | configuration-json)>
      </load-configuration>
</rpc>
```

5. Accepts the tag stream emitted by the Junos XML protocol server in response to each request and extracts its contents, as described in "Parse the Junos XML Protocol Server Response" on page 87.

   The server returns the <rpc-reply> and the <load-configuration-results> elements in response to a load operation. If the load operation is successful, the <load-configuration-results> element encloses the <load-success/> tag.

```
<rpc-reply xmlns:junos="URL">
    <load-configuration-results>
        <load-success/>
    </load-configuration-results>
</rpc-reply>
```

   If the load operation fails, the <load-configuration-results> element instead encloses the <load-error-count> element, which indicates the number of errors that occurred. The reply also includes one or more <xnm:error> elements with information about each of the errors. In this case, the application or administrator must address the errors before committing the configuration.

```
<rpc-reply xmlns:junos="URL">
    <load-configuration-results>
        <xnm:error>
            <!-- error information -->
        </xnm:error>
        <load-error-count>count</load-error-count>
```

```
    </load-configuration-results>
</rpc-reply>
```

6. (Optional) Verifies the syntactic correctness of the candidate configuration or a private copy before committing it. See "Verify Configuration Syntax Using the Junos XML Protocol" on page 287.

7. Commits the configuration.

   To commit the candidate configuration or a private copy, see "Commit the Candidate Configuration Using the Junos XML Protocol" on page 288.

   To commit an ephemeral database instance, see "Commit and Synchronize Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol" on page 339.

8. Closes or unlocks the target configuration database using one of the following options:

   - Unlocks the candidate configuration, if locked. Other users and applications cannot change the configuration while it remains locked.
   - Closes a private copy of the candidate configuration.
   - Closes an open instance of the ephemeral configuration database.

   For more information, see "Lock, Unlock, or Create a Private Copy of the Candidate Configuration Using the Junos XML Protocol" on page 94 and "Enable and Configure Instances of the Ephemeral Configuration Database" on page 328.

9. Ends the Junos XML protocol session and closes the connection to the device, as described in "End a Junos XML Protocol Session and Close the Connection" on page 100.

RELATED DOCUMENTATION

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Upload and Format Configuration Data in a Junos XML Protocol Session

A Junos XML protocol client application can load configuration data into a selected target configuration database on devices running Junos OS or devices running Junos OS Evolved. A client application can use the <load-configuration> operation with the appropriate attributes to specify the source and format of the configuration data to load. The application can provide new configuration data using a text file or

streaming data. The data format can be Junos XML elements, formatted ASCII text, configuration mode commands, or JavaScript Object Notation (JSON) data.

A client application can stream configuration changes within the session or reference a data file with the required configuration changes. Each method has advantages and disadvantages. Streaming data enables you to send your configuration data inline, using your Junos XML protocol connection. Streaming data 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; there is no need to include the possibly complex configuration data stream.

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

- "Upload Configuration Data as a File Using the Junos XML Protocol" on page 209
- "Upload Configuration Data as a Data Stream Using the Junos XML Protocol" on page 213
- "Define the Format of Configuration Data to Upload in a Junos XML Protocol Session" on page 215

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Replace the Configuration Using the Junos XML Protocol | 219

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Upload Configuration Data as a File Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can use the `<load-configuration>` operation to load configuration data onto a Junos device from a local or remote file.

A Junos XML protocol client application can load configuration data from a file onto a device running Junos OS or a device running Junos OS Evolved. The file can be a local file on the Junos device or a remote file on an FTP or HTTP server. After loading the configuration data, the client must commit the configuration to make it active.

Before loading the file, an administrator saves the configuration data as the contents of the file. The configuration data can be Junos XML elements, formatted ASCII text, Junos OS configuration mode commands, or JavaScript Object Notation (JSON). For more information, see "Define the Format of Configuration Data to Upload in a Junos XML Protocol Session" on page 215.

NOTE: When you load configuration data from a file, you do not need to enclose data formatted as ASCII text, configuration mode commands, or JSON in `<configuration-text>`, `<configuration-set>`, or `<configuration-json>` elements as you do for a data stream.

To load the configuration data from a local or remote file, the Junos XML protocol client application emits the `<rpc>` element and encloses the `<load-configuration/>` operation with the `url` attribute. The `format` and `action` attributes specify the format of the data and how to load the data into the target configuration database.

```
<rpc>
    <load-configuration url="url" format="format" action="action"/>
</rpc>
```

The value of the `url` attribute can be a local file path, an FTP location, or an HTTP URL.

- A local file can have one of the following forms:
  - ***/path/filename*** —File on a mounted file system, either on the local flash drive 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 file on an FTP server has the following form:

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

- A file 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 user's home directory. To specify an absolute path, start the path with the characters %2F; for example, `ftp://username:password@hostname/%2Fpath/filename` .

You can combine the `url` attribute with one or more of the following attributes in the `<load-configuration/>` tag:

- format
- action

The request must include the format attribute to specify the format of the data in the file unless the data format is Junos XML elements, which is the default. You can also include the action attribute to specify how to load the new data into the existing configuration. Table 10 on page 211 outlines the data format and corresponding attributes.

Table 10: load-configuration format and action Attributes

Data Format	format Attribute	action Attribute
ASCII text	format="text"	action="(merge | override | replace | update)"
Configuration mode commands	format="text"	action="set"
JSON	format="json"	action="(merge | override | update)"
Junos XML elements (default)	format="xml"	action="(merge | override | replace | update)"

For example, if the data is Junos XML tag elements, include the format="xml" attribute or omit the format attribute, which defaults to XML.

```
<rpc>
    <load-configuration url="file-location"/>
</rpc>
```

If the data is formatted ASCII text, include the format="text" attribute.

```
<rpc>
    <load-configuration url="file-location" format="text"/>
</rpc>
```

If the data comprises configuration mode commands, include the `action="set"` and `format="text"` attributes.

```
<rpc>
    <load-configuration url="file-location" action="set" format="text"/>
</rpc>
```

If the data uses JSON format, include the `format="json"` attribute.

```
<rpc>
    <load-configuration url="file-location" format="json"/>
</rpc>
```

The following example loads Junos XML-tagged configuration data stored in the file **/var/configs/user-accounts** from the FTP server cfg-server.mycompany.com. The opening `<load-configuration>` tag appears on two lines for legibility only.

```
<rpc>
    <load-configuration
       url="ftp://admin:AdminPwd@cfg-server.mycompany.com/var/configs/user-accounts"/>
</rpc>
```

The Junos XML Protocol server response is:

```
<rpc-reply xmlns:junos="url">
    <load-configuration-results>
        <load-success/>
    </load-configuration-results>
</rpc>
```

RELATED DOCUMENTATION

- Request Configuration Changes Using the Junos XML Protocol | 206
- Upload and Format Configuration Data in a Junos XML Protocol Session | 208
- Upload Configuration Data as a Data Stream Using the Junos XML Protocol | 213
- Define the Format of Configuration Data to Upload in a Junos XML Protocol Session | 215
- Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Upload Configuration Data as a Data Stream Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can use the <load-configuration> operation to load configuration data onto a Junos device as a data stream.

A Junos XML protocol client application can load configuration data as a data stream onto a device running Junos OS or a device running Junos OS Evolved. To load the configuration as a data stream, a client application emits the <rpc> element and the <load-configuration> element with the appropriate child tags. The configuration data format can be Junos XML elements, formatted ASCII text, Junos OS configuration mode commands, or JavaScript Object Notation (JSON).

To load configuration data as Junos XML elements, the application emits the elements representing all levels of the configuration hierarchy from the root (represented by the <configuration> element) down to each element to change.

```
<rpc>
    <load-configuration>
        <configuration>
            <!-- elements representing the configuration data -->
        </configuration>
    </load-configuration>
</rpc>
```

To load configuration data as formatted ASCII text, the application includes the format="text" attribute in the opening <load-configuration> tag. The application encloses the configuration statements in a <configuration-text> element.

```
<rpc>
    <load-configuration format="text">
        <configuration-text>
             /* formatted ASCII configuration data */
        </configuration-text>
    </load-configuration>
</rpc>
```

To load configuration data as Junos OS configuration mode commands, the application includes the `action="set"` and `format="text"` attributes in the opening `<load-configuration>` tag. The application encloses the configuration mode commands in a `<configuration-set>` element.

```
<rpc>
    <load-configuration action="set" format="text">
        <configuration-set>
            /* configuration mode commands */
        </configuration-set>
    </load-configuration>
</rpc>
```

To load configuration data using JSON format, the application includes the `format="json"` attribute in the opening `<load-configuration>` tag. The application encloses the JSON data in a `<configuration-json>` element.

```
<rpc>
    <load-configuration format="json">
        <configuration-json>
            /* JSON configuration data */
        </configuration-json>
    </load-configuration>
</rpc>
```

For information about the syntax for Junos XML elements, formatted ASCII text, configuration mode commands, and JSON format, see "Define the Format of Configuration Data to Upload in a Junos XML Protocol Session" on page 215.

RELATED DOCUMENTATION

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Upload Configuration Data as a File Using the Junos XML Protocol | 209

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Define the Format of Configuration Data to Upload in a Junos XML Protocol Session

SUMMARY

Junos XML protocol client applications can use the <load-configuration> operation with the appropriate attribute values to load configuration data in the requested format from a file or a data stream.

A Junos XML protocol client application can load configuration data on a device running Junos OS or a device running Junos OS Evolved. The application can load the data from a file or as a data stream. The configuration data can use any of the following formats:

- Junos XML elements (default)
- Formatted ASCII text
- Configuration mode commands
- JavaScript Object Notation (JSON)

To specify the format of the data, a client application includes the `format` attribute in the opening <load-configuration> tag. The <load-configuration> operation can combine the `format` attribute with the following attributes:

- `url`
- `action`

> **NOTE**: JSON format only supports `action` values of merge, override, and update.

The following sections discuss how to load configuration data from a file or as a data stream using the requested format in a Junos XML protocol session.

Junos XML Elements

When providing Junos XML data, the application includes the elements representing all levels of the configuration hierarchy from the root (the <configuration> element) down to each new or changed element.

```
<configuration>
    <!-- elements representing the configuration data -->
</configuration>
```

In the request, the application can include the format="xml" attribute or omit the format attribute in the <load-configuration> tag, since this is the default. To load the configuration data from a file, the application also includes the url attribute. To load the configuration data as a data stream, the application sends the data in the request.

```
<rpc>
    <load-configuration url="file-location"/>
</rpc>

<rpc>
    <load-configuration>
        <configuration>
            <!-- elements representing the configuration data -->
        </configuration>
    </load-configuration>
</rpc>
```

Formatted ASCII Text

If the application provides the configuration data as formatted ASCII text, it uses the standard Junos OS CLI notation to indicate the hierarchical relationships between configuration statements. This notation uses the newline character, tabs and other white space, braces, and square brackets. For each new or changed element, the complete statement path is specified, starting with the top-level statement that appears directly under the [edit] hierarchy level.

When providing configuration data as ASCII text, the application must include the format="text" attribute in the <load-configuration> tag. To load the configuration data from a file, the application also includes the

`url` attribute. To load the configuration data as a data stream, the application encloses the data in a `<configuration-text>` element.

```
<rpc>
    <load-configuration url="file-location" format="text"/>
</rpc>

<rpc>
    <load-configuration format="text">
        <configuration-text>
            /* formatted ASCII configuration data */
        </configuration-text>
    </load-configuration>
</rpc>
```

Configuration Mode Commands

A client application can load configuration data as configuration mode commands. The device executes the configuration instructions line by line. For each element, you can specify the complete statement path in the command, or you can use navigation commands, such as `edit` and `up`, to move around the configuration hierarchy as you would in CLI configuration mode.

When providing configuration mode commands, the application must include the `action="set"` and `format="text"` attributes in the `<load-configuration>` tag. To load the configuration data from a file, the application also includes the `url` attribute. To load the configuration data as a data stream, the application encloses the commands in a `<configuration-set>` element.

```
<rpc>
    <load-configuration url="file-location" action="set" format="text"/>
</rpc>

<rpc>
    <load-configuration action="set" format="text">
        <configuration-set>
            /* configuration mode commands to load */
        </configuration-set>
    </load-configuration>
</rpc>
```

JSON

A client application can load JSON configuration data onto a device. The application includes the configuration data representing all levels of the configuration hierarchy from the root configuration object down to each new or changed element.

When providing JSON configuration data, the application must include the `format="json"` attribute in the `<load-configuration>` tag. To load the configuration data from a file, the application also includes the `url` attribute. To load the configuration data as a data stream, the application encloses the data in a `<configuration-json>` element.

```
<rpc>
    <load-configuration url="file-location" format="json"/>
</rpc>

<rpc>
    <load-configuration format="json">
        <configuration-json>
            /* JSON-formatted configuration data */
        </configuration-json>
    </load-configuration>
</rpc>
```

RELATED DOCUMENTATION

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session

A Junos XML protocol client application can replace the entire configuration or change specific portions of the configuration on devices running Junos OS or devices running Junos OS Evolved. To change part or all of the configuration, the application uses the `<load-configuration>` operation and includes the appropriate attributes and child tag elements.

The elements to include depend on the action that the client application wants to perform. For information about how to modify the configuration, see the following topics:

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | **206**

Upload and Format Configuration Data in a Junos XML Protocol Session | **208**

Replace the Configuration Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can completely replace the configuration data in the candidate configuration, a private copy of the candidate configuration, or an ephemeral database instance.

IN THIS SECTION

A Junos XML protocol client application can completely replace the configuration on devices running Junos OS and devices running Junos OS Evolved. A client application can replace the configuration data in any of the following configuration databases:

- Candidate configuration
- Private copy of the candidate configuration

- Ephemeral database instance

A client application can replace the candidate configuration or a private copy of it by performing one of the following operations:

- Loading new configuration data
- Rolling back to a previously committed configuration
- Loading the rescue configuration

NOTE: The ephemeral configuration database does not support rolling back to a previous version of the configuration.

The following sections discuss how to replace all configuration data in the candidate configuration or open configuration database. After replacing the data, a client application must commit the configuration to make it the active configuration on the device. To modify individual configuration elements instead, see "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

Replacing the Candidate or Ephemeral Configuration with New Data

A client application can replace all configuration data in the candidate configuration or open configuration database with new configuration data. To replace the configuration, a client application executes the `<load-configuration>` operation and includes the `action="override"` attribute. If a client application issues the `<open-configuration>` operation to open a specific configuration database before executing the `<load-configuration>` operation, the server loads the configuration data into the open configuration database. Otherwise, the server loads the configuration data into the candidate configuration.

```
<rpc>
    <!-- For a file -->
        <load-configuration action="override" url="file" [format="format"]/>

    <!-- For a data stream -->
        <load-configuration action="override" [format="format"]>
            <!-- configuration data -->
        </load-configuration>
</rpc>
```

For more information about the `url` and `format` attributes and the syntax for the new configuration data, see "Upload and Format Configuration Data in a Junos XML Protocol Session" on page 208.

The following example replaces the entire candidate configuration with the contents of the file **/tmp/conf.xml**. The file contains Junos XML tag elements (the default), so this request omits the `format` attribute.

Client Application

```
<rpc>
  <load-configuration action="override" url="/tmp/conf.xml"/>
</rpc>
```

Junos XML Protocol Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

Rolling Back the Candidate Configuration to a Previously Committed Configuration

Junos OS and Junos OS Evolved store a copy of the most recently committed configuration and up to 49 previous configurations, depending on the platform. You can roll back to any of the stored configurations. Rolling back the configuration is useful when configuration changes cause undesirable results, and you want to revert to a known working configuration. The process is similar to making configuration changes on a device, but instead of loading configuration data, you perform a rollback. The rollback replaces the entire candidate configuration with a previously committed configuration.

When you successfully commit a configuration, Junos OS assigns that configuration a unique configuration revision identifier. The configuration is also associated with a rollback index, where the most recently committed configuration has rollback index 0. Whereas the rollback index for a previously committed configuration increments with each commit, the configuration revision ID remains static for the same configuration. When you roll back to a previously committed configuration, you can reference the configuration by its current rollback index or its configuration revision ID.

To display the rollback index and configuration revision ID for previously committed configurations, issue the `show system commit include-configuration-revision` operational mode command. For example, the following output displays four commits each with a rollback index and corresponding configuration revision identifier.

```
user@host> show system commit include-configuration-revision
0   2025-09-19 11:54:36 PDT by admin via cli re0-1758308074-4
1   2025-09-15 13:38:09 PDT by admin via cli re0-1757968687-3
```

```
2   2025-09-15 10:40:07 PDT by root via cli commit synchronize re0-1757958004-2
3   2025-09-15 10:39:15 PDT by root via other re0-1757957907-1
```

Table 11 on page 222 outlines the methods that a Junos XML protocol client can use to replace the candidate configuration (or a private copy) with a previously committed configuration. Each option references either the rollback index or the configuration revision identifier, as is appropriate for that method.

Table 11: Methods to Roll Back the Configuration in Junos XML Protocol Sessions

RPC	Description	Example
<load-configuration rollback="*index*"/>	Roll back to the configuration with the given rollback index.	<rpc> <load-configuration rollback="1"/> </rpc>
<load-configuration configuration-revision="*revision-id*"/>	Roll back to the configuration with the given configuration revision identifier.	<rpc> <load-configuration configuration-revision="re0-1757968687-3"/> </rpc>
<rollback-config> with <index>	Roll back to the configuration with the given rollback index. This RPC is useful when an application does not support executing RPCs that include XML attributes.	<rpc> <rollback-config> <index>1</index> </rollback-config> </rpc>

NOTE: Junos OS does not support rolling back the configuration committed to an instance of the ephemeral configuration database. Thus the ephemeral database does not support using the <rollback-config> RPC or the <load-configuration> operation with either the rollback or the configuration-revision attributes.

To use the <load-configuration> operation to replace the candidate configuration or a private copy of it with a previously committed configuration, a client application executes the <load-configuration/> operation and includes the rollback or configuration-revision attribute. The rollback value is the numerical rollback index of the appropriate previous configuration. Valid values are 0 (zero, for the most recently

committed configuration) through one less than the number of stored previous configurations. The configuration-revision value is the configuration revision ID of the configuration to load, for example, re0-1757968687-3.

```
<rpc>
    <load-configuration rollback="index"/>
</rpc>
```

```
<rpc>
    <load-configuration configuration-revision="revision-id"/>
</rpc>
```

The Junos XML protocol server indicates that the load operation was successful by returning the <load-configuration-results> and <load-success/> elements in its RPC reply.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.1R1/junos">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

To use the <rollback-config> RPC to load a previously committed configuration, a client application executes the <rollback-config> RPC with the <index> element. The <index> element specifies the rollback index for the configuration to load.

```
<rpc>
  <rollback-config>
    <index>1</index>
  </rollback-config>
</rpc>
```

The Junos XML protocol server indicates that the load operation was successful by returning the <rollback-config-results> and <load-success/> elements in its RPC reply.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.1R1/junos">
  <rollback-config-results>
    <load-success/>
```

```
  </rollback-config-results>
</rpc-reply>
```

If the load operation is successful, the client application must commit the configuration to make it the active configuration on the device. If the server encounters an error while loading the rollback configuration, it returns an <xnm:error> element with information about the error.

Replacing the Candidate Configuration with a Rescue Configuration

A rescue configuration allows you to define a known working configuration or a configuration with a known state that you can restore at any time. You use the rescue configuration to revert to a known configuration or as a last resort if the device configuration and the backup configuration files become damaged beyond repair. When you create a rescue configuration, the device saves the most recently committed configuration as the rescue configuration.

The rescue configuration must exist on the device before you can load it. To replace the candidate configuration or a private copy of it with the device's rescue configuration, a Junos XML protocol application can use one of the following methods:

- Execute the <load-configuration/> operation with the rescue="rescue" attribute.
- Execute the <rollback-config> RPC with the <rescue/> child element. This RPC is useful when an application does not support executing RPCs that include XML attributes.

To use the <load-configuration/> operation to replace the candidate configuration with the rescue configuration, include the rescue="rescue" attribute.

```
<rpc>
    <load-configuration rescue="rescue"/>
</rpc>
```

The Junos XML protocol server indicates that the load operation was successful by returning the <load-configuration-results> and <load-success/> elements in its RPC reply.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.1R1/junos">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

To use the <rollback-config> RPC to load the rescue configuration, a client application emits the <rollback-config> element and the <rescue/> child tag.

```
<rpc>
  <rollback-config>
    <rescue/>
  </rollback-config>
</rpc>
```

The Junos XML protocol server indicates that the load operation was successful by returning the <rollback-config-results> and <load-success/> elements in its RPC reply.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.1R1/junos">
  <rollback-config-results>
    <load-success/>
  </rollback-config-results>
</rpc-reply>
```

If the load operation is successful, the client application must commit the configuration to make it the active configuration on the device. If the rescue configuration does not exist or the server encounters another error while loading the configuration data, the server returns an <xnm:error> element with information about the error.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
18.1R1	Starting in Junos OS Release 18.1R1, a client application can replace all configuration data in an ephemeral configuration database instance with new data.
18.1R1	Starting in Junos OS Release 18.1R1, a client application can use the <rollback-config> RPC with the appropriate child tags to roll back to a previously committed configuration or load a rescue configuration.

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can create, modify, and delete configuration elements on Junos devices using different formats.

A Junos XML protocol client application can create, modify, or delete one or more configuration elements (hierarchy levels and configuration objects) on devices running Junos OS or devices running Junos OS Evolved. A client application can change the configuration elements in the candidate configuration or an open configuration database. The application can load the changes as Junos XML elements, formatted ASCII text, configuration mode commands, or JavaScript Object Notation (JSON).

For more information about the source and formatting for configuration elements, see "Upload and Format Configuration Data in a Junos XML Protocol Session" on page 208.

For information about the operations a client application can perform on configuration elements, see the following sections:

- "Create New Elements in Configuration Data Using the Junos XML Protocol" on page 229
- "Merge Elements in Configuration Data Using the Junos XML Protocol" on page 232
- "Replace Elements in Configuration Data Using the Junos XML Protocol" on page 237
- "Replace Only Updated Elements in Configuration Data Using the Junos XML Protocol" on page 241
- "Delete Elements in Configuration Data Using the Junos XML Protocol" on page 244
- "Rename Objects In Configuration Data Using the Junos XML Protocol" on page 255
- "Reorder Elements In Configuration Data Using the Junos XML Protocol" on page 258
- "Protect or Unprotect a Configuration Object Using the Junos XML Protocol" on page 263
- "Change a Configuration Element's Activation State Using the Junos XML Protocol" on page 268
- "Replace Patterns in Configuration Data Using the NETCONF or Junos XML Protocol" on page 282

The following sections provide an overview of how to create, modify, or delete configuration elements in the different formats.

Junos XML

To use Junos XML tag elements to represent an element, the application includes the tag elements representing all levels in the configuration hierarchy from the root (represented by the <configuration> element) down to the element's container tag element. The attributes and child tag elements that are included depend on the operation being performed on the element. The syntax applies both to the contents of a file and to a data stream. In the following example, the identifier tag element is called <name>:

```
<configuration>
   <!-- opening tag for each parent of the element -->
      <container-tag [operation-attribute="value"]>
          <name>identifier</name>  <!-- if the element has an identifier -->
          <!-- child tag elements --> <!-- if appropriate  -->
      </container-tag>
   <!-- closing tag for each parent of the element -->
</configuration>
```

Formatted ASCII Text

To use formatted ASCII text to represent an element, the application includes the complete statement path, starting with a statement that can appear directly under the [edit] hierarchy level. The attributes and child statements to include depend on the operation being performed on the element. The application encloses the set of statements in a <configuration-text> element when it uploads the configuration data as a data stream. The application omits the <configuration-text> element when the configuration data is stored in and loaded from a file.

```
<configuration-text>
    /* statements for parent levels of the element */
        operation-to-perform:      # if appropriate
        element identifier   {      # if the element has an identifier
            /* child statements */    # if appropriate for the operation
        }
    /* closing braces for parent levels of the element */
</configuration-text>
```

When loading formatted ASCII text, the application must include the format="text" attribute in the <load-configuration> tag.

Configuration Mode Commands

To use configuration mode commands to create, modify, or delete an element, the application includes the commands as they would be typed in configuration mode in the CLI. The device executes the configuration instructions line by line in the order provided. You can specify the complete statement path in the command, or you can use CLI navigation commands such as `edit` and `up`, to move around the configuration hierarchy.

The application encloses the set of commands in a `<configuration-set>` element when it uploads the configuration data as a data stream. The application omits the `<configuration-set>` element when the configuration data is stored in and loaded from a file.

```
<configuration-set>
     /* configuration mode commands */
</configuration-set>
```

When loading configuration mode commands, the application must include the `action="set"` and `format="text"` attributes in the `<load-configuration>` tag.

JSON

To use JSON format to represent an element, the application includes JSON objects representing all levels in the configuration hierarchy from the root down to the JSON object representing that element. The attributes and child objects to include depend on the operation being performed on the element. If the attribute value is a Boolean data type, do not enclose the value in quotes.

The application encloses the JSON data in a `<configuration-json>` element when it uploads the configuration data as a data stream. The application omits the `<configuration-json>` element when the configuration data is stored in and loaded from a file.

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels of the element */
            "container-tag" : {
                "@" : {
                    "operation-attribute" : ( "value" | boolean )
                },
                "object" : [
                {
                    "@" : {
                        "operation-attribute" : ( "value" | boolean )
```

```
                    },
                    "(name | element-identifier)" : "identifier",
                    "statement-name" : "statement-value",
                    "@statement-name" : {
                        "operation-attribute" : ( "value" | boolean )
                    },
                    /* additional JSON data and child objects */
                }
                ]
            }
        /* closing braces for parent levels of the element */
    }
}
</configuration-json>
```

When loading data in JSON format, the application must include the `format="json"` attribute in the `<load-configuration>` tag.

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Create New Elements in Configuration Data Using the Junos XML Protocol

A Junos XML protocol client application can create new configuration elements (hierarchy levels or configuration objects) on devices running Junos OS or devices running Junos OS Evolved. A client application first includes the basic tag elements, formatted ASCII statements, configuration mode commands, or JSON objects described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

For Junos XML elements and formatted ASCII text, you can create new elements in either merge mode or replace mode. For more information, see "Merge Elements in Configuration Data Using the Junos XML Protocol" on page 232 and "Replace Elements in Configuration Data Using the Junos XML Protocol" on page 237. In replace mode, the application includes the `action="replace"` attribute in the `<load-configuration>` tag. When you use JavaScript Object Notation (JSON), you must use merge mode to create new objects.

The following sections outline how to create new configuration items using the different formats:

Junos XML

To use Junos XML to create the element, an application includes each of the element's identifier tags (if it has them) and all child tag elements that it is defining for the element. In the following example, the identifier tag is <name>. The application does not need to include any attributes in the opening container tag for the new element.

```
<configuration>
    <!-- opening tag for each parent of the element -->
        <container-tag>
            <name>identifier</name>
                <!-- tag elements for other children, if any -->
         </container-tag>
    <!-- closing tag for each parent of the element -->
</configuration>
```

Formatted ASCII Text

To use formatted ASCII text to create the element, an application includes each of the element's identifiers (if it has them) and all child statements (with values if appropriate) that it is defining for the element. It does not need to include an operator before the new element. To provide the configuration data as a data stream, enclose the data in a <configuration-text> element.

```
<configuration-text>
    /* statements for parent levels of the element */
        element identifier {
            /* child statements if any */
        }
    /* closing braces for parent levels of the element */
</configuration-text>
```

Configuration Mode Commands

To use configuration mode commands to create new elements, an application includes the set command as you would execute it in the CLI. The command includes the statement path to the element, the element's identifier if it has one, and all child statements (with values if appropriate) that it is defining for

the element. To provide the configuration data as a data stream, enclose the commands in a <configuration-set> element.

```
<configuration-set>
    set statement-path-to-element element identifier child-elements
</configuration-set>
```

JSON

To use JSON to create the object, an application includes the object's identifiers (if it has them) and all data and child objects that it is defining for the object. The application does not need to include any specific operation attributes to create the new object. To provide the configuration data as a data stream, enclose the data in a <configuration-json> element.

In the following example, the object's identifier is the field designated as "name".

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels of the element */
            "container-tag" : {
                "object" : [
                {
                    "name" : "identifier",
                    /* data and child objects */  # if any
                }
                ],
                /* data and child objects */  # if any
            }
        /* closing braces for parent levels of the element */
    }
}
</configuration-json>
```

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Merge Elements in Configuration Data Using the Junos XML Protocol

By default, the Junos XML protocol server *merges* loaded configuration data into the candidate configuration according to the following rules. (The rules also apply to a private copy of the configuration or an open instance of the ephemeral configuration database, but for simplicity the following discussion refers to the candidate configuration only.)

- A configuration element (hierarchy level or configuration object) that exists in the candidate but not in the loaded configuration remains unchanged.
- A configuration element that exists in the loaded configuration but not in the candidate is added to the candidate.
- If a configuration element exists in both configurations, the semantics are as follows:
  - If a child statement of the configuration element (represented by a child tag element) exists in the candidate but not in the loaded configuration, it remains unchanged.
  - If a child statement exists in the loaded configuration but not in the candidate, it is added to the candidate.
  - If a child statement exists in both configurations, the value in the loaded configuration replaces the value in the candidate.

Merge mode is the default mode for new configuration elements. To merge new configuration data with the existing configuration, the application emits the `<load-configuration>` operation in an `<rpc>` element.

```
<rpc>
    <!-- For a file -->
        <load-configuration url="file" [format="format"]/>

    <!-- For a data stream -->
        <load-configuration [format="format"]>
            <!-- configuration data -->
        </load-configuration>
</rpc>
```

To explicitly specify merge mode for configuration data that uses Junos XML elements, formatted ASCII text, or JSON format, the application can include the `action="merge"` attribute in the `<load-configuration>` tag, as shown in the later examples.

For more information about the `url` and `format` attributes, see "Upload and Format Configuration Data in a Junos XML Protocol Session" on page 208.

The following sections outline how to merge configuration items into the existing configuration using the different formats:

Junos XML

To use Junos XML to merge the element into the configuration, the application first includes the basic tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. The application does not include any attributes in the element's container tag. To add or change the value of a child element, the application includes the elements for it. If a child remains unchanged, the loaded configuration does not need to include it. In the following example, the identifier tag is `<name>`:

```
<configuration>
    <!-- opening tag for each parent of the element -->
        <container-tag>
            <name>identifier</name> <!-- if the element has an identifier -->
            <!-- tag elements for other children, if any -->
        </container-tag>
    <!-- closing tag for each parent of the element -->
</configuration>
```

The following example uses Junos XML elements (the default) to merge configuration data for interface ge-0/0/1 at the `[edit interfaces]` hierarchy level.

```
<rpc>
  <load-configuration action="merge">
    <configuration>
      <interfaces>
        <interface>
          <name>ge-0/0/1</name>
          <unit>
            <name>0</name>
            <family>
              <inet>
                <address>
                  <name>10.0.0.1/8</name>
                </address>
              </inet>
```

```
            </family>
          </unit>
        </interface>
      </interfaces>
    </configuration>
  </load-configuration>
</rpc>
```

Formatted ASCII Text

To use formatted ASCII text to merge statements, the application first includes the statement path described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. The application does not include a preceding operator, but it does include the element's identifier if it has one. To add or change the value of a child element, the application includes the elements for it. If a child remains unchanged, the loaded configuration does not need to include it. To provide the configuration data as a data stream, enclose the data in a <configuration-text> element.

```
<configuration-text>
/* statements for parent levels of the element */
    element identifier  {
          /* child statements if any */
    }
/* closing braces for parent levels of the element */
</configuration-text>
```

The following example uses formatted ASCII text to merge the configuration data for interface ge-0/0/1.

```
<rpc>
  <load-configuration action="merge" format="text">
    <configuration-text>
    interfaces {
      ge-0/0/1 {
          unit 0 {
              family inet {
                  address 10.0.0.1/8;
              }
          }
      }
    }
```

```
    </configuration-text>
  </load-configuration>
</rpc>
```

Configuration Mode Commands

To use configuration mode commands to merge new elements, an application includes the `set` command, the statement path to the element, and the element's identifier if it has one. To add or change the value of a child element, the application includes the child elements or statements in the command. If a child remains unchanged, the configuration mode commands do not need to include it. To provide the configuration data as a data stream, enclose the commands in a `<configuration-set>` element.

```
<configuration-set>
    set statement-path-to-element element identifier child-statements
</configuration-set>
```

To load configuration mode commands, the application includes the `action="set"` and `format="text"` attributes in the `<load-configuration>` tag. The following example uses configuration mode commands to define the ge-0/0/1 interface.

```
<rpc>
  <load-configuration action="set" format="text">
    <configuration-set>
      set interfaces ge-0/0/1 unit 0 family inet address 10.0.0.1/8
    </configuration-set>
  </load-configuration>
</rpc>
```

JSON

To use JSON data to merge elements into the configuration, an application first includes the basic JSON data described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. The application does not need to include any specific operation attributes in the JSON configuration data in order to merge the new or changed element. To add or change the value of a child element, the application includes the JSON data or child objects for it. If a child remains unchanged, the loaded configuration does not need to include it. To provide the configuration data as a data stream, enclose the data in a `<configuration-json>` element.

In the following example, the JSON member that specifies the element's identifier has the field name "name":

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels of the element */
            "container-tag" : {
                "object" : [
                {
                    "name" : "identifier",
                    "statement-name" : "statement-value",   # if any
                    /* additional data and child objects */   # if any
                }
                ],
                /* data and child objects */   # if any
            }
        /* closing braces for parent levels of the element */
    }
}
</configuration-json>
```

The following example uses JSON configuration data to define the ge-0/0/1 interface.

```
<rpc>
<load-configuration format="json">
<configuration-json>
{
   "configuration" : {
      "interfaces" : {
         "interface" : [
         {
            "name" : "ge-0/0/1",
            "unit" : [
            {
               "name" : 0,
               "family" : {
                  "inet" : {
                     "address" : [
                     {
                        "name" : "10.0.0.1/8"
```

```
                    }
                    ]
                }
            }
        }
        ]
    }
    ]
  }
 }
}
</configuration-json>
</load-configuration>
</rpc>
```

RELATED DOCUMENTATION

- Request Configuration Changes Using the Junos XML Protocol | 206
- Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Replace Elements in Configuration Data Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use the `<load-configuration>` operation with the `replace` action to replace individual configuration elements, including hierarchy levels or configuration objects.

A Junos XML protocol client application can replace individual configuration elements (hierarchy levels or configuration objects) on a device running Junos OS or a device running Junos OS Evolved. To replace

individual elements, a client application emits the <load-configuration> element with the action="replace" attribute in an <rpc> element.

```
<rpc>
    <!-- For a file -->
        <load-configuration action="replace" url="file" [format=("xml"|"text")]/>

    <!-- For a data stream -->
        <load-configuration action="replace" [format=("xml"|"text")]>
            <!-- configuration data -->
        </load-configuration>
</rpc>
```

NOTE: Starting in Junos OS Release 18.1R1, the ephemeral configuration database supports loading configuration data using the <load-configuration> action attribute values of override and replace in addition to the previously supported values of merge and set.

NOTE: Junos OS does not support using the replace operation when loading JSON configuration data. To replace configuration elements when using JSON, you must delete the existing element and then add the replacement element.

To perform replace operations, a client application must load configuration data as Junos XML elements or formatted ASCII text. The configuration data must specify which objects to replace. The following sections discuss how to replace configuration elements using the different formats.

Junos XML

To use Junos XML elements to define the data to replace, the application first includes the basic tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. The application includes the replace="replace" attribute in the opening container tag of the object to replace. Within the container tag, the application includes the same child tag elements as for a new element: each of the replacement's identifier tag elements (if it has them) and all child tag elements being defined for the replacement element.

In the following example, the application replaces the object that has the replace="replace" attribute and the identifier specified in the <name> element.

```
<configuration>
    <!-- opening tag for each parent of the element -->
```

```
        <container-tag replace="replace">
            <name>identifier</name>
            <!-- tag elements for other children, if any -->
        </container-tag>
    <!-- closing tag for each parent of the element -->
</configuration>
```

The following example replaces the configuration for the `admin` class at the `[edit system login class]` hierarchy level. The configuration data uses Junos XML-tagged format (the default).

```
<rpc>
<load-configuration action="replace">
  <configuration>
    <system>
      <login>
        <class replace="replace">
          <name>admin</name>
          <permissions>configure</permissions>
          <permissions>admin</permissions>
          <permissions>admin-control</permissions>
        </class>
      </login>
    </system>
  </configuration>
</load-configuration>
</rpc>
```

Formatted ASCII Text

To use formatted ASCII text to replace the element, the application first includes the complete statement path described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. For the replacement element, the application includes each of the replacement's identifiers (if it has them) and all child statements (with values if appropriate) that it is defining. The request includes the `replace:` statement on the line that just precedes the element's container statement. To provide the configuration data as a data stream, enclose the data in a `<configuration-text>` element.

```
<configuration-text>
    /* statements for parent levels of the element */
        replace:
        element  identifier {
```

```
            /* child statements if any */
        }
    /* closing braces for parent levels of the element */
</configuration-text>
```

The following example uses formatted ASCII text to replace the configuration for the `admin` class at the `[edit system login class]` hierarchy level.

```
<rpc>
<load-configuration action="replace" format="text">
  <configuration-text>
    system {
      login {
        replace:
        class admin {
          permissions [configure admin admin-control]
        }
      }
    }
  </configuration-text>
</load-configuration>
</rpc>
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
18.1R1	Starting in Junos OS Release 18.1R1, the ephemeral configuration database supports loading configuration data using the `<load-configuration>` `action` attribute values of `override` and `replace` in addition to the previously supported values of `merge` and `set`.

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Replace Only Updated Elements in Configuration Data Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use the <load-configuration> operation with the update action to load a complete configuration but replace only the updated configuration elements.

A Junos XML protocol client application can load a complete configuration but replace just the updated configuration elements (hierarchy levels and configuration objects) on devices running Junos or devices running Junos OS Evolved. To replace configuration elements only if the elements differ between the loaded configuration and the existing configuration, the application emits the <rpc> element and encloses the <load-configuration> operation with the action="update" attribute.

```
<rpc>
    <!-- For a file -->
        <load-configuration action="update" url="file" [format="format"]/>

    <!-- For a data stream -->
        <load-configuration action="update" [format="format"]>
            <!-- configuration data -->
        </load-configuration>
</rpc>
```

For more information about the url and format attributes, see "Upload and Format Configuration Data in a Junos XML Protocol Session" on page 208.

NOTE: Starting in Junos OS Release 21.1R1, the ephemeral configuration database supports the action="update" attribute on supported platforms.

The action="update" operation is equivalent to the load update configuration mode command in the Junos OS CLI. When you use a load update operation, you provide a complete configuration, and the device compares the two configurations. Each configuration element that is different in the loaded configuration replaces its corresponding element in the existing configuration. Elements that are the same in both configurations remain unchanged. When you commit the configuration, only system processes that are affected by the changed configuration elements parse the new configuration.

To define the replacement elements, the application uses the same syntax as for new elements. For more information, see "Create New Elements in Configuration Data Using the Junos XML Protocol" on page 229. The following sections outline how to update the configuration using the specified format.

Junos XML

To use Junos XML elements to represent the replacement elements, an application includes each of the element's identifier tags (if it has them) and all child tag elements that it is defining for the element. In the following Junos XML example, the object's identifier is called `name`.

```
<configuration>
    <!-- opening tag for each parent of the element -->
        <container-tag>
            <name>identifier</name>
            <!-- tag elements for other children, if any -->
        </container-tag>
    <!-- closing tag for each parent of the element -->
</configuration>
```

The following example updates the candidate configuration with the contents of the local file **/tmp/conf.xml**. The file contains a complete configuration represented as Junos XML elements (the default), so the request omits the `format` attribute.

```
<rpc>
    <load-configuration action="update" url="/tmp/conf.xml"/>
</rpc>
```

Formatted ASCII Text

To use formatted ASCII text to represent the data, an application includes each of the element's identifiers (if it has them) and all child statements (with values if appropriate) that it is defining for the element. To provide the configuration data as a data stream, enclose the data in a `<configuration-text>` element.

```
<configuration-text>
    /* statements for parent levels of the element */
        element identifier {
            /* child statements if any */
        }
```

```
    /* closing braces for parent levels of the element */
</configuration-text>
```

JSON

To use Junos XML elements to represent the replacement objects, an application includes the object's identifiers (if it has them) and all data and child objects that it is defining for the object. To provide the configuration data as a data stream, enclose the data in a <configuration-json> element.

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels of the element */
            "container-tag" : {
                "object" : [
                {
                    "name" : "identifier",
                    "statement-name" : "statement-value",   # if any
                    /* additional data and child objects */   # if any
                }
                ],
                /* data and child objects */   # if any
            }
        /* closing braces for parent levels of  the element */
    }
}
</configuration-json>
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
21.1R1	Starting in Junos OS Release 21.1R1, the ephemeral configuration database supports the action="update" attribute on supported platforms.

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Delete Elements in Configuration Data Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can delete configuration elements from the selected configuration database on Junos devices.

IN THIS SECTION

Junos XML protocol client applications can delete configuration elements (hierarchy levels or configuration objects) from the candidate configuration or open configuration database on devices running Junos OS or devices running Junos OS Evolved. To delete elements, a client application first emits the basic tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. A client application then includes the appropriate attribute or operator for the chosen format.

Table 12 on page 244 summarizes how a client application deletes configuration objects using the different data formats. The placement of the attribute or operator depends on the type of element to delete.

Table 12: Delete Configuration Objects

Format	Method to Delete Object
Junos XML	Include the `delete="delete"` attribute in the opening tag for each element to delete.
Formatted ASCII text	Include the `delete:` operator on the line that just precedes the statement to delete.

Table 12: Delete Configuration Objects *(Continued)*

Format	Method to Delete Object
Configuration mode commands	Use the `delete` command and specify the path to the element.
JSON	Include the `"operation" : "delete"` attribute in the attribute list for each object to delete.

To provide the configuration data as a data stream, enclose formatted ASCII text, configuration mode commands, or JSON data in a `<configuration-text>`, `<configuration-set>`, or `<configuration-json>` element, respectively. To load the data from a file, you do not need to include any additional elements.

Delete a Hierarchy Level or Container Object

A client application can delete a hierarchy level and all of its children (or a container object that has children but no identifier). To delete a hierarchy level, a client application first includes the basic tag elements or configuration statements for its parent levels, as described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

To use Junos XML elements to delete a hierarchy level, an application includes the `delete="delete"` attribute in the empty tag that represents the hierarchy level or container object.

```
<configuration>
    <!-- opening tag for each parent level -->
        <level-or-object delete="delete"/>
    <!-- closing tag for each parent level -->
</configuration>
```

To use formatted ASCII text, an application places the `delete:` statement on the line that precedes the hierarchy level to remove. The application also places a semicolon after the level to remove (even though the existing configuration uses curly braces that enclose its child statements).

```
    /* statements for parent levels */
        delete:
        object-or-level;
    /* closing braces for parent levels */
```

To use configuration mode commands, an application specifies the `delete` command and the statement path to the hierarchy level or object to remove.

```
delete statement-path-to-level-or-object
```

To use JSON configuration data, an application includes the `"operation" : "delete"` attribute in the attribute list for the hierarchy or container object to remove.

```
{
    "configuration" : {
        /* JSON objects for parent levels */
            "object-or-level" : {
                "@" : {
                    "operation" : "delete"
                }
            }
        /* closing braces for parent levels */
    }
}
```

The following example removes the `[edit protocols ospf]` hierarchy level from the candidate configuration using Junos XML tag elements:

```
<rpc>
  <load-configuration>
    <configuration>
      <protocols>
        <ospf delete="delete"/>
      </protocols>
    </configuration>
  </load-configuration>
</rpc>
```

The following example removes the `[edit protocols ospf]` hierarchy level from the candidate configuration using JSON configuration data. The request provides the configuration data as a data stream, so the application encloses the data in a `<configuration-json>` element.

```
<rpc>
  <load-configuration format="json">
    <configuration-json>
```

```
    {
        "configuration" : {
            "protocols" : {
                "ospf" : {
                  "@" : {
                      "operation" : "delete"
                  }
                }
            }
        }
    }
    </configuration-json>
  </load-configuration>
</rpc>
```

Delete a Configuration Object That Has an Identifier

To delete a configuration object that has an identifier, a client application first includes the basic tag elements or configuration statements for its parent levels, as described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

To Junos XML elements to delete the object, an application includes the `delete="delete"` attribute in the opening tag for the object to remove. In the object's container element, the application encloses only the identifier element, not the elements that represent any other characteristics of the object. In the following example, the `<name>` element is the object's identifier:

```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object delete="delete">
            <name>identifier</name>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

NOTE: The `delete` attribute appears in the opening container tag, not in the identifier tag. 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.

To use formatted ASCII text to delete the object, an application places the `delete:` statement on the line that precedes the object and its identifier.

```
    /* statements for parent levels of the object */
        delete:
        object identifier;
    /* closing braces for parent levels of the object */
```

To use configuration mode commands, an application specifies the `delete` command, the statement path to the object, and the object and its identifier.

```
delete statement-path-to-object object identifier
```

To use JSON configuration data, an application includes the `"operation" : "delete"` attribute in the attribute list for the object. In the container object, it encloses only the name/value pair representing the identifier. In the following example, the "name" field identifies the object to remove:

```
{
    "configuration" : {
        /* JSON objects for parent levels of the object */
            "object" : [
            {
                "@" : {
                    "operation" : "delete"
                },
                "name" : "identifier"
            }
            ]
        /* closing braces for parent levels of  the object */
    }
}
```

The following example uses Junos XML tag elements to remove the user object `barbara` from the `[edit system login user]` hierarchy level in the candidate configuration.

```
<rpc>
  <load-configuration>
    <configuration>
      <system>
        <login>
```

```
          <user delete="delete">
            <name>barbara</name>
          </user>
        </login>
      </system>
    </configuration>
  </load-configuration>
</rpc>
```

The following example uses JSON configuration data to remove the user object `barbara` from the `[edit system login user]` hierarchy level in the candidate configuration. The request provides the configuration data as a data stream, so the application encloses the data in a `<configuration-json>` element.

```
<rpc>
<load-configuration format="json">
<configuration-json>
{
    "configuration" : {
        "system" : {
            "login" : {
                "user" : [
                {
                    "@" : {
                        "operation" : "delete"
                    },
                    "name" : "barbara"
                }
                ]
            }
        }
    }
}
</configuration-json>
</load-configuration>
</rpc>
```

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

A client application can delete either a fixed-form option or an option that takes just one value from a configuration object. The client application first includes the basic tag elements or configuration statements for its parent levels, as described in "Create, Modify, or Delete Configuration Elements Using

the Junos XML Protocol" on page 226. For information about deleting an option that can take multiple values, see "Delete Values from a Multivalue Option of a Configuration Object" on page 252.

To use Junos XML tag elements to delete the option, an application includes the `delete="delete"` attribute in the empty tag for that option. The application does not include tag elements for children that should remain in the configuration. In the following example, the `<name>` element is the object's identifier:

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

When using formatted ASCII text, the application places the `delete:` statement above each option:

```
/* statements for parent levels of the object */
    object identifier;
    delete:
        option1;
    delete:
        option2;
/* closing braces for parent levels of the object */
```

When using configuration mode commands, the application specifies the `delete` command, the statement path to the option, and the option to be removed. The commands can specify the full path to the option statement. Alternatively, the commands can navigate to the hierarchy level of the object and delete the option statement from that location. Use a separate command to delete each option.

```
    delete statement-path-to-object object identifier option1
    delete statement-path-to-object object identifier option2
```

```
    edit statement-path-to-object object identifier
    delete option1
    delete option2
```

When using JSON configuration data to delete an option, the application includes the `"operation" : "delete"` attribute in the option's attribute list. To delete options for a hierarchy level or container object, specify the options to delete at that level.

```
{
    "configuration" : {
        /* JSON objects for parent levels  */
            "level-or-object" : {
                "@option1" : {
                    "operation" : "delete"
                },
                "@option2" : {
                    "operation" : "delete"
                }
            }
        /* closing braces for parent levels */
    }
}
```

To delete options for an object that has an identifier, include the identifier first, and then specify the options to delete. In the following example, the element's identifier has the field name "name":

```
{
    "configuration" : {
        /* JSON objects for parent levels of the object */
            "object" : [
            {
                "name" : "identifier",
                "@option1" : {
                    "operation" : "delete"
                },
                "@option2" : {
                    "operation" : "delete"
                }
            }
            ]
        /* closing braces for parent levels of the object */
    }
}
```

The following example removes the fixed-form option `disable` at the `[edit forwarding-options sampling]` hierarchy level using Junos XML tag elements.

```
<rpc>
  <load-configuration>
    <configuration>
      <forwarding-options>
        <sampling>
          <disable delete="delete"/>
        </sampling>
      </forwarding-options>
    </configuration>
  </load-configuration>
</rpc>
```

Delete Values from a Multivalue Option of a Configuration Object

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 element for each value. In the following example, the `<name>` element is the identifier:

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

For example, consider the permissions of a login class, which are displayed as a list in brackets.

```
class admin {
    permissions [ admin admin-control configure ];
}
```

Junos XML format lists each permission in a separate <permissions> element.

```
<configuration>
    <system>
        <login>
            <class>
                <name>admin</name>
                <permissions>admin</permissions>
                <permissions>admin-control</permissions>
                <permissions>configure</permissions>
            </class>
        </login>
    </system>
</configuration>
```

To remove one or more values for such an object, a client application first includes the basic tag elements or configuration statements for its parent levels, as described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

When using Junos XML elements to remove a value, the application includes the delete="delete" attribute in the opening tag for each value. It does not include elements that represent values to be retained. In the following example, the parent object's identifier is <name>:

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

When using formatted ASCII text, the application repeats the parent statement for each value and places the delete: statement above each paired statement and value:

```
    /* statements for parent levels of the parent object */
        parent-object identifier;
        delete:
            object value1;
        delete:
```

```
            object value2;
    /* closing braces for parent levels of the parent object */
```

When using configuration mode commands, the application specifies the `delete` command, the statement path to each value, and the value to be removed. The commands can specify the full path to the value. Alternatively, the commands can navigate to the hierarchy level of the object and delete the value from that location. Use a separate command to delete each value.

```
    delete statement-path-to-parent-object parent-object identifier object value1
    delete statement-path-to-parent-object parent-object identifier object value2
```

```
    edit statement-path-to-parent-object parent-object identifier object
    delete value1
    delete value2
```

In JSON, a multivalue option is a name/value pair where the field name is the object name, and its value, which represents the options, is an array of strings. Junos OS does not support using JSON to delete single values from an object with a multivalue option. To update the option list, you must delete the existing object and then configure a new object with the required set of values.

The following example uses Junos XML elements to remove two of the permissions granted to the `user-accounts` login class.

```
<rpc>
<load-configuration>
  <configuration>
    <system>
      <login>
        <class>
          <name>user-accounts</name>
          <permissions delete="delete">configure</permissions>
          <permissions delete="delete">control</permissions>
        </class>
      </login>
    </system>
  </configuration>
</load-configuration>
</rpc>
```

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Rename Objects In Configuration Data Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can rename configuration objects on Junos devices.

A Junos XML protocol client application can change the name of one or more of a configuration object's identifiers. To rename the object, a client application first includes the tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226.

To use Junos XML elements to rename an object, the client application includes the object and its identifier element with the current name. In the object's opening tag, the application includes:

- The `rename="rename"` attribute.
- An attribute with the same name as the identifier tag name. For most objects, this is `name`. The value for this attribute is the new identifier for the object.

```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object rename="rename" name="new-name">
            <name>current-name</name>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

If the object has multiple identifiers, the application includes both an identifier element and an attribute in the opening tag for each one. If one or more of the identifiers is not changing, set its attribute value to its current name. For example:

```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object rename="rename" changing-identifier="new-name" \
                    unchanging-identifier="current-name">
            <changing-identifier>current-name</changing-identifier>
            <unchanging-identifier>current-name</unchanging-identifier>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

To use configuration mode commands to rename an object, the application specifies the `rename` command equivalent to the CLI configuration mode command. If the object has multiple identifiers, the application includes a separate `rename` command for each identifier.

```
<configuration-set>
    rename statement-path-to-object object current-name to object new-name
</configuration-set>
```

NOTE: You cannot use the rename operation when you use formatted ASCII text or JSON to represent the configuration data.

The following example changes the name of a firewall filter from `access-control` to `new-access-control` using Junos XML elements.

```
<rpc>
  <load-configuration>
    <configuration>
        <firewall>
            <family>
                <inet>
                    <filter rename="rename" name="new-access-control">
                        <name>access-control</name>
                    </filter>
                </inet>
            </family>
```

```
        </firewall>
    </configuration>
  </load-configuration>
</rpc>
```

The operation is equivalent to the following configuration mode command:

```
[edit firewall family inet]
user@host# rename filter access-control to filter new-access-control
```

The following example changes the name of a firewall filter from `access-control` to `new-access-control` using configuration mode commands:

```
<rpc>
    <load-configuration action="set" format="text">
        <configuration-set>
            rename firewall family inet filter access-control to filter new-access-control
        </configuration-set>
    </load-configuration>
</rpc>
```

The following example shows how to change the identifiers for an OSPF virtual link (defined at the `[edit protocols ospf area area]` hierarchy level) from `neighbor-id 192.168.0.3` and `transit-area 10.10.10.1` to `neighbor-id 192.168.0.7` and `transit-area 10.10.10.5`.

```
[edit protocols ospf area area]
user@host# rename virtual-link neighbor-id 192.168.0.3 transit-area 10.10.10.1 to virtual-link
neighbor-id 192.168.0.7 transit-area 10.10.10.5
```

```
<rpc>
   <load-configuration>
      <configuration>
         <protocols>
            <ospf>
               <area>
                  <name>area</name>
                  <virtual-link rename="rename" neighbor-id="192.168.0.7" transit-
area="10.10.10.5">
```

```
                    <neighbor-id>192.168.0.3</neighbor-id>
                    <transit-area>10.10.10.1</transit-area>
                  </virtual-link>
                </area>
              </ospf>
            </protocols>
          </configuration>
        </load-configuration>
</rpc>
```

RELATED DOCUMENTATION

- Request Configuration Changes Using the Junos XML Protocol | 206
- Upload and Format Configuration Data in a Junos XML Protocol Session | 208
- Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Reorder Elements In Configuration Data Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can use the `insert` attribute to reorder configuration objects that use an ordered set.

Junos devices store and display most configuration objects in predetermined positions in the configuration hierarchy. Thus, the order in which the system creates the object or its children is not significant. However, some configuration objects—such as routing policies and firewall filters—consist of elements that the system must process and analyze sequentially in order to produce the intended routing behavior. When a Junos XML protocol client application adds a new element to an ordered set, by default, the system appends the element to the existing list of elements. The client application can reorder the elements, if appropriate.

A Junos XML protocol client application can change the order of configuration elements in an ordered set by including the `insert` attribute in the request. A client application first includes the tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. When using Junos XML format, the application emits the tag or container tag that represents the

ordered set. The application also encloses the tag for each identifier (if it has them). In the following examples, the identifier tag is <name>.

To add or move an element to the first position in an ordered set, the application includes the insert="first" attribute in the element's opening tag. The following example inserts an object at the beginning of the ordered list defined in the given leaf statement:

```
<configuration>
    <!-- opening tag for each parent of the set -->
        <ordered-set insert="first">identifier-for-moving-object</ordered-set>
    <!-- closing tag for each parent of the set -->
</configuration>
```

The following example inserts the object identified by the <name> element at the beginning of the list:

```
<configuration>
    <!-- opening tag for each parent of the set -->
        <ordered-set insert="first">
            <name>identifier-for-moving-object</name>
        </ordered-set>
    <!-- closing tag for each parent of the set -->
</configuration>
```

A client application can also add or move an element to a position that is relative to another element. The application includes the insert="before" or insert="after" attribute in the opening tag to indicate the new position of the moving element relative to another reference element in the set. To identify the reference element, the application includes each of the reference element's identifiers as an attribute in the opening tag for the ordered set.

In the following example, the leaf statement contains an ordered list. The application inserts the new list item before or after the item identified by the name="*referent-value*" attribute.

```
<configuration>
    <!-- opening tag for each parent of the set -->
        <ordered-set insert="(before | after)" name="referent-value">identifier-for-moving-
object</ordered-set>
    <!-- closing tag for each parent of the set -->
</configuration>
```

In the following example, the elements in the set have one identifier called <name>:

```
<configuration>
    <!-- opening tag for each parent of the set -->
        <ordered-set insert="(before | after)" name="referent-value">
            <name>identifier-for-moving-object</name>
        </ordered-set>
    <!-- closing tag for each parent of the set -->
</configuration>
```

In the following example, each element in the set has two identifiers. The opening tag appears on two lines for legibility only.

```
<configuration>
    <!-- opening tag for each parent of the set -->
        <ordered-set insert="(before | after)" identifier1="referent-value"
          identifier2="referent-value">
            <identifier1>value-for-moving-object</identifier1>
            <identifier2>value-for-moving-object</identifier2>
        </ordered-set>
    <!-- closing tag for each parent of the set -->
</configuration>
```

When using configuration mode commands to reorder elements, the application specifies the insert command that is equivalent to the CLI configuration mode command.

```
<configuration-set>
    insert statement-path-to-object identifier-for-moving-object (before | after) referent-value
</configuration-set>
```

NOTE: The insert="first" attribute has no equivalent CLI configuration mode command.

NOTE: The reordering operation does not support using formatted ASCII text or JSON to represent the configuration data.

The following example moves an existing firewall filter term called `default` and places it after the filter called `rule1` in the firewall configuration for `filter1`.

```
<rpc>
  <load-configuration>
    <configuration>
      <firewall>
        <family>
          <inet>
            <filter>
              <name>filter1</name>
              <term insert="after" name="rule1">
                 <name>default</name>
              </term>
            </filter>
          </inet>
        </family>
      </firewall>
    </configuration>
  </load-configuration>
</rpc>
```

The following example performs the same operation as in the previous example but uses the equivalent configuration mode command:

```
<rpc>
    <load-configuration action="set" format="text">
        <configuration-set>
            insert firewall family inet filter filter1 term default after term rule1
        </configuration-set>
    </load-configuration>
</rpc>
```

In the following example, the BGP group `my_group` initially has an export policy list with one export policy `deny_all`. The RPC inserts a second export policy named `my_export_policy` before the `deny_all` export policy.

```
<rpc>
  <load-configuration>
  <configuration>
    <protocols>
```

```
      <bgp>
        <group>
          <name>my_group</name>
          <export insert="before" name="deny_all">my_export_policy</export>
         </group>
       </bgp>
     </protocols>
  </configuration>
  </load-configuration>
</rpc>
```

The following example moves an OSPF virtual link defined at the [edit protocols ospf area *area*] hierarchy level. The example moves the link with identifiers neighbor-id 192.168.0.3 and transit-area 10.10.10.1 before the link with identifiers neighbor-id 192.168.0.5 and transit-area 10.10.10.2. This operation is equivalent to the following configuration mode command:

```
[edit protocols ospf area area]
user@host# insert virtual-link neighbor-id 192.168.0.3 transit-area 10.10.10.1 before virtual-
link neighbor-id 192.168.0.5 transit-area 10.10.10.2
```

Client Application

```
<rpc>
<load-configuration>
    <configuration>
        <protocols>
            <ospf>
                <area>
                    <name>area</name>
                    <virtual-link insert="before" neighbor-id="192.168.0.5" transit-
area="10.10.10.2">
                        <neighbor-id>192.168.0.3</neighbor-id>
                        <transit-area>10.10.10.1</transit-area>
                    </virtual-link>
                </area>
            </ospf>
        </protocols>
    </configuration>
</load-configuration>
</rpc>
```

RELATED DOCUMENTATION

Request Configuration Changes Using the Junos XML Protocol | 206

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226

Protect or Unprotect a Configuration Object Using the Junos XML Protocol

The `protect` attribute prevents changes to selected configuration hierarchies and statements. You cannot alter a protected element either manually through the CLI or automatically using commit scripts or remote procedure calls. If you attempt to make configuration changes to a protected statement or within a protected hierarchy, the device issues a warning, and the configuration change fails.

If a configuration hierarchy or statement is protected, users cannot perform the following activities:

- Delete or modify the hierarchy or a statement or identifier within the protected hierarchy (Deletion of an unprotected hierarchy that contains protected elements deletes all unprotected child elements and preserves all protected child elements.)
- Insert a new configuration statement or an identifier within the protected hierarchy
- Rename the protected statement or a statement or identifier within the protected hierarchy
- Copy a configuration into the protected hierarchy
- Activate or deactivate the protected statements or statements within the protected hierarchy
- Annotate the protected statement or hierarchy, or statements within the protected hierarchy

If you protect a configuration statement or hierarchy that does not exist, the device first creates the configuration element and then protects it. If you unprotect a statement or element that is not protected, no action is taken.

You can identify protected elements when you display the configuration. Table 13 on page 264 describes how you identify protected elements for configuration data in different formats.

Table 13: Identifying Protected Elements

Format	Identifier
Configuration mode commands	The `protect` commands indicate protected elements.
JSON	Protected hierarchies and statements include the `"protect" : true` attribute in their attribute list.
Text	Protected elements are preceded by `protect:`.
XML	The opening tag of the protected element contains the `protect="protect"` attribute.

NOTE: A user or client application must have permission to modify the configuration in order to protect or unprotect configuration objects.

In a NETCONF or Junos XML protocol session with a Junos device, to protect a configuration element from changes or to unprotect a previously protected element, a client application first includes the tag elements described in "Creating, Modifying, or Deleting Configuration Elements Using the Junos XML Protocol" on page 226.

XML

When using Junos XML tag elements to represent the configuration, the client application includes the `protect="protect"` or `unprotect="unprotect"` attribute in the object's opening tag. The application includes any necessary identifier tag element. In the following sample RPC, the identifier tag element is called `<name>`:

```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object (protect="protect" | unprotect="unprotect")>
            <name>identifier</name>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

NOTE: In a YANG-compliant NETCONF session, you can protect or unprotect a configuration element in XML data by including the `xmlns:jcmd="http://yang.juniper.net/junos/jcmd"` and `jcmd:protect="(true | false)"` attributes in the tag element. For more information, see *YANG Metadata Annotations for Junos Devices*.

Text

When using formatted ASCII text to protect or unprotect an object, the application precedes the element with the `protect:` or `unprotect:` operator as appropriate. If you are protecting a hierarchy level and no additional child elements under that hierarchy, add a semicolon after the element statement.

```
<configuration-text>
    /* statements for parent levels */

        /* For an object with an identifier */
            (protect: | unprotect:)
            object identifier {
                 /* Child configuration statements */
            }

        /* For a hierarchy level or object without an identifier */
            (protect: | unprotect:)
            element {
                /* Child configuration statements */
            }

    /* closing braces for parent levels */
</configuration-text>
```

Configuration Mode (Set) Commands

When using configuration mode commands to protect an object, the application specifies the `protect` or `unprotect` command equivalent to the CLI configuration mode command. You can protect both hierarchies and individual statements.

```
<configuration-set>
    (protect | unprotect) statement-path-to-hierarchy
    (protect | unprotect) statement-path-to-object object identifier
</configuration-set>
```

JSON

When using JSON configuration data to represent the configuration, the client application protects or unprotects an object by including the appropriate attribute in the attribute list of the object. The client includes the `"protect" : true` attribute to protect the object and includes either the `"protect" : false` or `"unprotect" : true` attribute to unprotect the object. To protect or unprotect an object that has an identifier, the client also includes the identifier for the object.

The following generic JSON configuration indicates the placement of the attribute when protecting a hierarchy, an object that has an identifier, and a leaf statement.

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent hierarchies */
            "hierarchy" : {
                "@" : {
                    "comment" : "/* protect a hierarchy */" ,
                    "protect" : true
                },
                "object" : [
                {
                    "@" : {
                        "comment" : "/* protect an object with an identifier */" ,
                        "protect" : true
                    },
                    "name" : "identifier",
                    "@statement-name" : {
                        "comment" : "/* protect a statement */" ,
                        "protect" : true
                    }
                }
                ]
            }
        /* closing braces for parent hierarchies */
    }
}
</configuration-json>
```

NOTE: In Junos OS configuration data that is represented using JSON, the value for the `"protect"` and `"unprotect"` attribute is type Boolean, which is expressed in lowercase and is not enclosed in quotes.

NOTE: In a YANG-compliant NETCONF session, you can protect or unprotect a configuration object in JSON data by including the `"junos-configuration-metadata:protect" : (true | false)` annotation in the statement's metadata object. For more information, see *YANG Metadata Annotations for Junos Devices*.

The following example protects the `[edit access]` hierarchy level of the configuration using Junos XML tag elements:

```
<rpc>
     <load-configuration>
          <configuration>
               <access protect="protect"/>
          </configuration>
     </load-configuration>
</rpc>
```

Once protected, any attempt to modify the `[edit access]` hierarchy level produces a warning. The following RPC attempts to delete the `[edit access]` hierarchy level. Because that hierarchy level is protected, the server returns a warning that the hierarchy is protected, and the configuration change fails.

```
<rpc>
     <load-configuration>
          <configuration>
               <access delete="delete"/>
          </configuration>
     </load-configuration>
</rpc>

<xnm:warning xmlns="http://xml.juniper.net/xnm/1.1/xnm" xmlns:xnm="http://
xml.juniper.net/xnm/1.1/xnm">
  <message>
    [access] is protected, 'access' cannot be deleted
```

```
  </message>
</xnm:warning>
```

RELATED DOCUMENTATION

protect

Request Configuration Changes Using the Junos XML Protocol | 206

Upload and Format Configuration Data in a Junos XML Protocol Session | 208

Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218

Change a Configuration Element's Activation State Using the Junos XML Protocol

IN THIS SECTION

When a configuration element (hierarchy level or configuration object) is deactivated and the configuration is committed, the deactivated element remains in the configuration, but the element does not affect the functioning of the device. Deactivating configuration elements is useful when you want to troubleshoot issues by suppressing the behavior of a configuration element without deleting it from the configuration. Additionally, you can configure and deactivate new configuration elements to prepare the configuration to accommodate new hardware before it is available.

In a NETCONF or Junos XML protocol session with a Junos device, a client application can deactivate an existing element or simultaneously create and deactivate a new element. A client application can also activate a deactivated element so that when the configuration is committed, the element again has an effect on the functioning of the device. The following sections discuss how to create and deactivate new configuration elements and how to activate or deactivate existing elements:

Deactivating a Newly Created Element

To create an element and immediately deactivate it, a client application first includes the basic tag elements or configuration statements for the new element and any child elements as described in "Creating, Modifying, or Deleting Configuration Elements Using the Junos XML Protocol" on page 226.

XML

When using Junos XML tag elements to create and deactivate a new element, the application includes the `inactive="inactive"` attribute in the opening tag for the new element. In the following example, the identifier tag element is called `<name>`:

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

NOTE: In a YANG-compliant NETCONF session, you can activate or deactivate a configuration element in XML data by including the `xmlns:jcmd="http://yang.juniper.net/junos/jcmd"` and `jcmd:active="(true | false)"` attributes in the tag element. For more information, see *YANG Metadata Annotations for Junos Devices*.

Text

When using formatted ASCII text to create and deactivate a new element, the application precedes the new element with the `inactive:` operator.

```
<configuration-text>
/* statements for parent levels */

    /* For an object with an identifier */
        inactive:
        object identifier {
            /* Child configuration statements */
        }

    /* For a hierarchy level or object without an identifier */
```

```
        inactive:
        element {
            /* Child configuration statements */
        }

/* closing braces for parent levels */
</configuration-text>
```

Configuration Mode (Set) Commands

When using configuration mode commands to create an inactive element, the application first creates the element with the `set` command and then deactivates it by using the `deactivate` command.

```
<configuration-set>
    set statement-path-to-object object identifier
    deactivate statement-path-to-object object identifier
</configuration-set>
```

JSON

When using JSON configuration data to create and deactivate a new element, the client application includes the `"inactive" : true` attribute in the attribute list for that element. The following generic JSON configuration indicates the placement of the attribute for deactivating a hierarchy or container object, an object that has an identifier, and a leaf statement.

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels */
            "level-or-container" : {
                "@" : {
                    "comment" : "/* deactivate a hierarchy */",
                    "inactive" : true
                },
                "object" : [
                {
                    "@" : {
                        "comment" : "/* deactivate an object with an identifier */",
                        "inactive" : true
                    },
                    "name" : "identifier",
                    "statement-name" : "statement-value",
```

```
                    "@statement-name" : {
                        "comment" : "/* deactivate a statement */",
                        "inactive" : true
                    },
                    /* additional data and child objects */  # if any
                }
                ]
            }
        /* closing braces for parent levels */
    }
}
</configuration-json>
```

NOTE: In a YANG-compliant NETCONF session, you can activate or deactivate a configuration object in JSON data by including the `"junos-configuration-metadata:active" : (true | false)` annotation in the statement's metdata object. For more information, see *YANG Metadata Annotations for Junos Devices*.

Deactivating or Reactivating an Existing Element

To deactivate an existing element, or activate a previously deactivated element, a client application includes the basic tag elements or configuration statements for its parent levels, as described in "Creating, Modifying, or Deleting Configuration Elements Using the Junos XML Protocol" on page 226.

XML

When using Junos XML tag elements to represent a configuration object that has an identifier, the application includes the `inactive="inactive"` or `active="active"` attribute in the object's opening container tag and also emits the identifier tag element and value. In the following example, the identifier tag element is called `<name>`. To represent a hierarchy level or container object that has children but does not have an identifier, the application uses an empty tag:

```
<configuration>
    <!-- opening tag for each parent of the element -->
        <!- - For an object with an identifier -->
        <object ( inactive="inactive" | active="active" ) >
            <name>identifier</name>
        </object>

        <!-- For a hierarchy level or object without an identifier -->
        <level-or-container ( inactive="inactive" | active="active" ) />
```

```
    <!-- closing tag for each parent of the element -->
</configuration>
```

NOTE: In a YANG-compliant NETCONF session, you can activate or deactivate a configuration element in XML data by including the `xmlns:jcmd="http://yang.juniper.net/junos/jcmd"` and `jcmd:active="(true | false)"` attributes in the tag element. For more information, see *YANG Metadata Annotations for Junos Devices*.

Text

When using formatted ASCII text to represent the element, the application precedes the element with the `inactive:` or `active:` operator. The name of a hierarchy level or container object is followed by a semicolon (even though in the existing configuration it is followed by curly braces that enclose its child statements):

```
<configuration-text>
/* statements for parent levels */

    /* For an object with an identifier */
        (inactive | active):
        object identifier;

    /* For a hierarchy level or object without an identifier */
        (inactive | active):
        object-or-level;

/* closing braces for parent levels */
</configuration-text>
```

Configuration Mode (Set) Commands

When using configuration mode commands to activate or deactivate an object, the application specifies the `activate` or `deactivate` command equivalent to the CLI configuration mode command.

```
<configuration-set>
    /* For an object with an identifier */
    activate statement-path-to-object object identifier
    deactivate statement-path-to-object object identifier

    /* For a hierarchy level or object without an identifier */
    activate statement-path-to-object-or-level object-or-level
```

```
    deactivate statement-path-to-object-or-level object-or-level
</configuration-set>
```

JSON

When using JSON to represent the element, the client application activates or deactivates the element by including the `"active" : true` or `"inactive" : true` attribute, respectively, in the attribute list of that element. The following generic JSON configuration indicates the placement of the attribute for activating or deactivating existing hierarchies or container objects, objects that have an identifier, and leaf statements.

```
<configuration-json>
{
    "configuration" : {
        /* JSON objects for parent levels */
            "level-or-container" : {
                "@" : {
                    "comment" : "/* activate or deactivate a hierarchy */",
                    "(active | inactive)" : true
                },
                "object" : [
                {
                    "@" : {
                        "comment" : "/* activate or deactivate an object with an identifier */",
                        "(active | inactive)" : true
                    },
                    "name" : "identifier",
                    "@statement-name" : {
                        "comment" : "/* activate or deactivate a statement */",
                        "(active | inactive)" : true
                    }
                }
                ]
            }
        /* closing braces for parent levels */
    }
}
</configuration-json>
```

NOTE: In a YANG-compliant NETCONF session, you can activate or deactivate a configuration object in JSON data by including the `"junos-configuration-metadata:active" : (true | false)` annotation in the statement's metadata object. For more information, see *YANG Metadata Annotations for Junos Devices*.

The following example shows how to deactivate the `isis` hierarchy level at the `[edit protocols]` hierarchy level in the candidate configuration using Junos XML tag elements.

Client Application

```
<rpc>
  <load-configuration>
    <configuration>
      <protocols>
        <isis inactive="inactive"/>
      </protocols>
    </configuration>
  </load-configuration>
</rpc>
```

Junos XML Protocol Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T11145

The following example shows how to deactivate the `isis` hierarchy level at the `[edit protocols]` hierarchy level in the candidate configuration using JSON.

```
<rpc>
<load-configuration format="json">
<configuration-json>
{
   "configuration" : {
      "protocols" : {
         "isis" : {
            "@" : {
               "inactive" : true
            }
         }
      }
   }
}
```

```
</configuration-json>
</load-configuration>
</rpc>
```

RELATED DOCUMENTATION

- Request Configuration Changes Using the Junos XML Protocol | 206
- Upload and Format Configuration Data in a Junos XML Protocol Session | 208
- Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | 218
- Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | 226
- Change a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol | 275

Change a Configuration Element's Activation State Simultaneously with Other Changes Using the Junos XML Protocol

IN THIS SECTION

In a Junos XML protocol session with a device running Junos OS, a client application can deactivate or reactivate an element at the same time it performs other operations on it (except deletion), by combining the appropriate attributes or operators with the `inactive` or `active` attribute or operator. For basic information about activating or deactivating an element, see "Changing a Configuration Element's Activation State Using the Junos XML Protocol" on page 268.

To define the element to deactivate or activate, a client application includes the basic tag elements or configuration statements for its parent levels, as described in "Creating, Modifying, or Deleting Configuration Elements Using the Junos XML Protocol" on page 226. When using Junos XML tag elements to represent the element, the application includes the `inactive="inactive"` or `active="active"`

attribute along with the other appropriate attributes in the <load-configuration> tag. When using formatted ASCII text, the application combines the inactive or active operator with the other operator.

For instructions, see the following sections:

Replacing an Element and Setting Its Activation State

To replace (completely reconfigure) an element and simultaneously deactivate or activate it, a client application includes the tag elements or statements that represent all of the element's characteristics (for complete information about the syntax for defining elements, see "Replacing Elements in Configuration Data Using the Junos XML Protocol" on page 237). The client application uses the attributes and operators discussed in the following examples to indicate which element is being replaced and activated or deactivated.

Using Junos XML Tag Elements for the Replacement Element

If using Junos XML tag elements to represent the element, a client application includes the action="replace" attribute in the <load-configuration> tag element:

```
<rpc>
    <!-- For a file -->
        <load-configuration action="replace" url="file"/>

    <!-- For a data stream -->
        <load-configuration action="replace">
            <!-- Junos XML tag elements -->
        </load-configuration>
</rpc>
```

In the opening tag for the replacement element, the application includes two attributes—the replace="replace" attribute and either the inactive="inactive" or active="active" attribute. It includes tag elements for all children being defined for the element. In the following, the identifier tag element is called <name>:

```
<configuration>
    <!-- opening tag for each parent of the element -->
        <element replace="replace" (inactive="inactive" | active="active")>
            <name>identifier</name> <!-- if element has an identifier -->
            <!-- tag elements for each child of the element -->
        </element>
    <!-- closing tag for each parent of the element -->
</configuration>
```

Using Formatted ASCII Text for the Replacement Element

If using formatted ASCII text to represent the element, a client application includes the `action="replace"` and `format="text"` attributes in the `<load-configuration>` tag:

```
<rpc>
    <!-- For a file -->
        <load-configuration action="replace" format="text" url="file"/>

    <!-- For a data stream -->
        <load-configuration action="replace" format="text">
            <!-- formatted ASCII configuration statements -->
        </load-configuration>
</rpc>
```

The application places the `inactive:` or `active:` operator on the line above the new element and the `replace:` operator directly before the new element. It includes all child statements that it is defining for all children of the element:

```
<configuration-text>
/* statements for parent levels */

    /* For an object with an identifier */
        (inactive | active):
        replace: object identifier {
            /* Child configuration statements */
        }

    /* For a hierarchy level or object without an identifier */
        (inactive | active):
        replace: element {
            /* Child configuration statements */
        }

/* closing braces for parent levels */
</configuration-text>
```

Reordering an Element and Setting Its Activation State

To reorder an element in an ordered list and simultaneously deactivate or activate it, the application combines the `insert` attribute and identifier attribute for the reference element with the `inactive` or `active`

attribute. In the following, the identifier tag element for the moving element is called <name>. The opening tag appears on two lines for legibility only:

```
<configuration>
   <!-- opening tag for each parent of the set -->
       <ordered-set insert="( before | after )" reference-identifier="value"
                   (inactive="inactive" | active="active")>
           <name>identifier-for-moving-object</name>
       </ordered-set>
   <!-- closing tag for each parent of the set -->
</configuration>
```

NOTE: The reordering operation is not available when formatted ASCII text is used to represent the configuration data.

For complete information about reordering elements, see "Reordering Elements In Configuration Data Using the Junos XML Protocol" on page 258.

Renaming an Object and Setting Its Activation State

To rename an object (change the value of one or more of its identifiers) and simultaneously deactivate or activate it, the application combines the rename attribute and identifier attribute for the new name with the inactive or active attribute.

If the object has one identifier (here called <name>), the syntax is as follows (the opening tag appears on two lines for legibility only):

```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object rename="rename" name="new-name" \
                (inactive="inactive" | active="active")>
            <name>current-name</name>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

If the object has multiple identifiers and only one is changing, the syntax is as follows (the opening tag appears on multiple lines for legibility only):

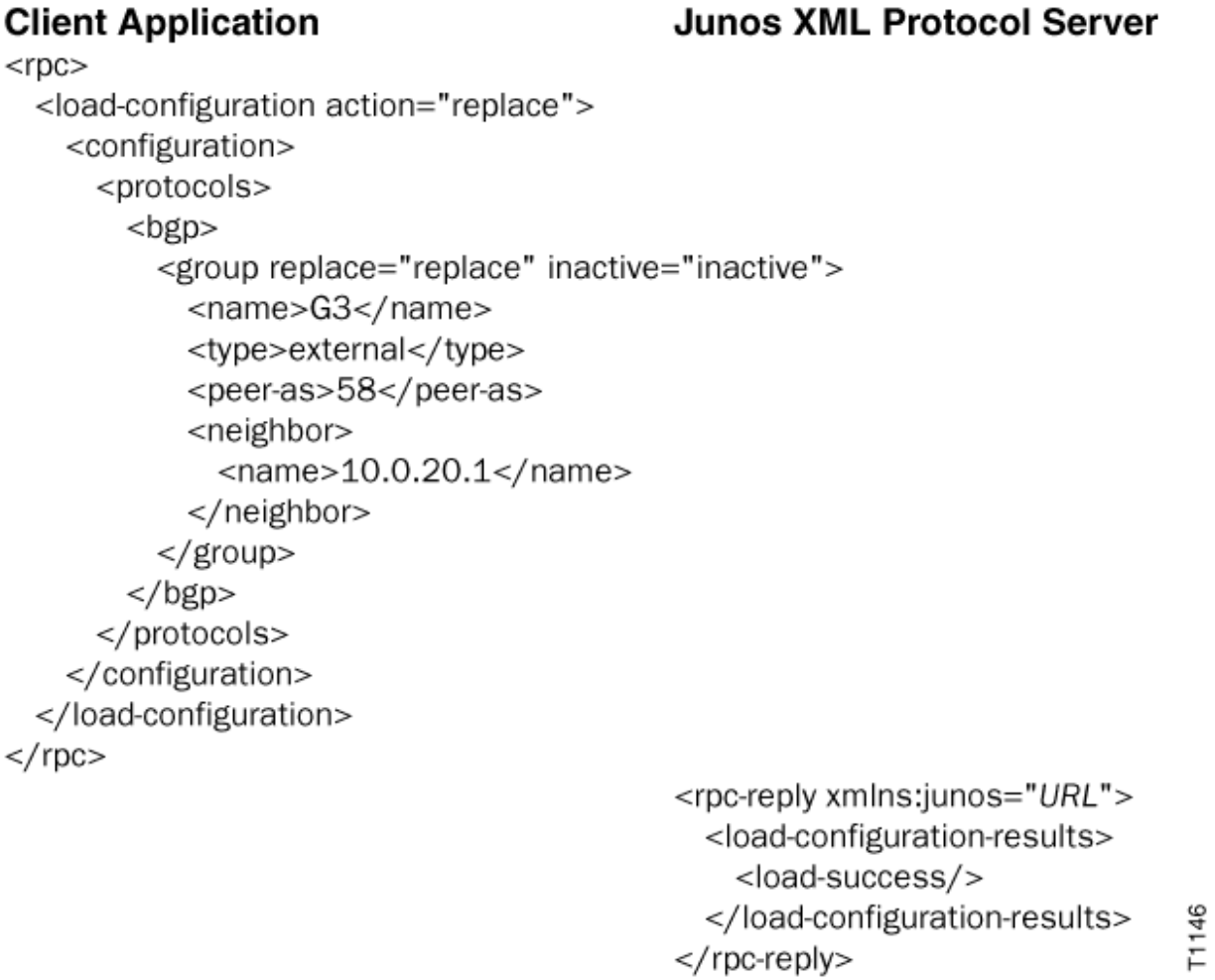
```
<configuration>
    <!-- opening tag for each parent of the object -->
        <object rename="rename" changing-identifier="new-name" \
                  unchanging-identifier="current-name" \
                    (inactive="inactive" | active="active")>
            <changing-identifier>current-name</changing-identifier>
            <unchanging-identifier>current-name</unchanging-identifier>
        </object>
    <!-- closing tag for each parent of the object -->
</configuration>
```

NOTE: The renaming operation is not available when formatted ASCII text is used to represent the configuration data.

For complete information about renaming elements, see "Renaming Objects In Configuration Data Using the Junos XML Protocol" on page 255.

Example: Replacing an Object and Deactivating It

The following example shows how to replace the information at the `[edit protocols bgp]` hierarchy level in the candidate configuration for the group called `G3`, and also deactivate the group so that it is not used in the actual configuration when the candidate is committed:



The following example shows how to use formatted ASCII text to make the same changes:

Client Application **Junos XML Protocol Server**

```
<rpc>
  <load-configuration action="replace" format="text">
    <configuration-text>
      protocols {
        bgp {
        replace:
          inactive: group G3 {
            type external;
            peer-as 58;
            neighbor 10.0.20.1;
          }
        }
      }
    </configuration-text>
  </load-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T11147

RELATED DOCUMENTATION

- Request Configuration Changes Using the Junos XML Protocol | **206**
- Upload and Format Configuration Data in a Junos XML Protocol Session | **208**
- Specify the Scope of Configuration Data to Upload in a Junos XML Protocol Session | **218**
- Replace the Configuration Using the Junos XML Protocol | **219**
- Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol | **226**
- Change a Configuration Element's Activation State Using the Junos XML Protocol | **268**

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

SUMMARY

NETCONF and Junos XML protocol client applications can replace variables and identifiers in the configuration when performing a <load-configuration> operation.

IN THIS SECTION

A NETCONF or Junos XML protocol client application can replace variables and identifiers in the configuration of devices running Junos OS or devices running Junos OS Evolved. For example, you might need to replace all occurrences of an interface name when a PIC is moved to another slot in the router.

To replace a pattern, a client application uses the <load-configuration> operation with the replace-pattern attribute. The replace-pattern attribute replaces the existing pattern with the new pattern. The scope of the replacement can be global or at a specified hierarchy or object level in the configuration. The functionality of the replace-pattern attribute is identical to that of the replace pattern configuration mode command in the Junos OS CLI.

NOTE: To use the replace pattern operations, you must use Junos XML elements for the configuration data format.

To replace a pattern, a client application emits the <rpc> and <load-configuration> elements and includes the basic Junos XML tag elements described in "Create, Modify, or Delete Configuration Elements Using the Junos XML Protocol" on page 226. At the hierarchy or object level where you want to replace the pattern, include the following attributes:

- replace-pattern—Pattern to replace.
- with—Replacement pattern.
- upto—(Optional) Number of occurrences to replace. If you omit this attribute or set it to zero, the device replaces all instances of the pattern within the specified scope.

The placement of the attributes within the configuration determines the scope of the replacement as described in the following sections.

Replace Patterns Globally Within the Configuration

To replace a pattern globally throughout the candidate configuration or open configuration database, include the `replace-pattern` and `with` attributes in the opening `<configuration>` tag. You can optionally include the `up-to` attribute to replace only a specified number of occurrences.

```
<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 172.16.1.1:

```
<rpc>
   <load-configuration>
      <configuration replace-pattern="172.17.1.5" with="172.16.1.1"/>
      </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/24.4R1/junos">
<configuration-information>
<configuration-output>
[edit groups global system ntp]
-    boot-server 172.17.1.5;
+    boot-server 172.16.1.1;
[edit groups global system ntp]
+     server 172.16.1.1;
-     server 172.17.1.5;
</configuration-output>
```

```
</configuration-information>
</rpc-reply>
```

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

A client application can replace a pattern under a specific hierarchy level including all of its children (or a container object that has children but no identifier). To replace the pattern within a specific hierarchy level, 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>
```

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/24.4R1/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>
```

Replace 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. Within the container tag, the application also includes the identifier element for that object. In the following example, the identifier tag 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>
```

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/24.4R1/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 | 198

Modifying the Configuration for a Device

Modifying the Configuration for a Device

replace

CHAPTER 12

Commit the Configuration on a Device Using the Junos XML Protocol

IN THIS CHAPTER

Verify Configuration Syntax Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, during the process of committing the candidate configuration or a private copy, the Junos XML protocol server first confirms that the candidate configuration is syntactically correct. If the syntax check fails, the server does not commit the 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.

To verify the syntax of the candidate configuration prior to committing it, a client application encloses an empty `<check/>` tag in the `<commit-configuration>` and `<rpc>` tag elements.

```
<rpc>
    <commit-configuration>
        <check/>
    </commit-configuration>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <commit-results>, and <routing-engine> tag elements. If the candidate configuration syntax is valid, the <routing-engine> tag element encloses the <commit-check-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the check succeeded (re0 on routing platforms that use a single Routing Engine, and either re0 or re1 on routing platforms that can have two Routing Engines).

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>
            <name>(re0 | re1)</name>
            <commit-check-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

If the candidate configuration syntax is not valid, the server returns an <xnm:error> element, which encloses child tag elements that describe the error.

The <check/> tag can be combined with the <synchronize/> tag to verify the check the syntactic correctness of a local configuration on both Routing Engines.

RELATED DOCUMENTATION

- Commit the Candidate Configuration Using the Junos XML Protocol | 288
- Commit a Private Copy of the Configuration Using the Junos XML Protocol | 290
- <commit-configuration> | 115

Commit the Candidate Configuration Using the Junos XML Protocol

When you commit the candidate configuration on a device running Junos OS, it becomes the active configuration on the routing, switching, or security platform. In a Junos XML protocol session, to commit the candidate configuration, a client application encloses the empty <commit-configuration/> tag in an <rpc> tag element.

```
<rpc>
    <commit-configuration/>
</rpc>
```

We recommend that the client application lock the candidate configuration before modifying it and emit the <commit-configuration/> 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. For instructions, see"Locking and Unlocking the Candidate Configuration or Creating a Private Copy Using the Junos XML Protocol" on page 94.

The Junos XML protocol server reports the results of the commit operation in <rpc-reply>, <commit-results>, and <routing-engine> tag elements. If the commit operation succeeds, the <routing-engine> tag element encloses the <commit-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the commit operation succeeded (re0 on devices that use a single Routing Engine, and either re0 or re1 on devices that can have two Routing Engines).

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>
            <name>(re0 | re1)</name>
            <commit-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

If the commit operation fails, the server returns an <xnm:error> tag element, which encloses child tag elements that describe the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration.

RELATED DOCUMENTATION

- Verify Configuration Syntax Using the Junos XML Protocol | 287
- Commit a Private Copy of the Configuration Using the Junos XML Protocol | 290
- Commit a Configuration at a Specified Time Using the Junos XML Protocol | 292
- Commit the Candidate Configuration Only After Confirmation Using the Junos XML Protocol | 294
- Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | 298
- Log a Message About a Commit Operation Using the Junos XML Protocol | 305
- <commit-configuration> | 115

Commit a Private Copy of the Configuration Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, to commit a private copy of the configuration so that it becomes the active configuration on the routing, switching, or security platform, a client application encloses the empty <commit-configuration/> tag in an <rpc> tag element (just as for the candidate configuration).

```
<rpc>
    <commit-configuration/>
</rpc>
```

The Junos XML protocol server creates a copy of the current candidate configuration, merges in the changes made to the private copy of the configuration, and then commits the combined candidate to make it the active configuration on the device. The server reports the results of the commit operation in <rpc-reply> and <commit-results> tag elements.

If the private copy does not include any changes, the server emits the opening <commit-results> tag and closing </commit-results> tags with nothing between.

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
    </commit-results>
</rpc-reply>
```

If the private copy includes changes and the commit operation succeeds, the server emits the <load-success/> tag when it merges the changes in the private copy into the candidate configuration. The <routing-engine> element encloses the <commit-success/> tag and the <name> element, which reports the name of the Routing Engine on which the commit operation succeeded (re0 on devices that use a single Routing Engine, and either re0 or re1 on devices that can have two Routing Engines).

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <load-success/>
        <routing-engine>
            <name>(re0 | re1)</name>
            <commit-success/>
        </routing-engine>
```

```
    </commit-results>
</rpc-reply>
```

If the private copy includes changes that conflict with the regular candidate configuration, the commit fails. The <load-error-count> tag element reports the number of errors and an <xnm:error> tag element encloses tag elements that describe the error.

There are restrictions on committing a private copy. For example, the commit 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.

Most of the variants of the commit operation are available for a private copy.

- Scheduling the commit for a later time, as described in "Committing a Configuration at a Specified Time Using the Junos XML Protocol" on page 292.
- Synchronizing the configuration on both Routing Engines, as described in "Committing and Synchronizing a Configuration on Redundant Control Planes Using the Junos XML Protocol" on page 298.
- Logging a commit-time message, as described in "Logging a Message About a Commit Operation Using the Junos XML Protocol" on page 305.

NOTE: The confirmed-commit operation is not available for a private copy. For information about using that operation for the regular candidate configuration, see "Committing the Candidate Configuration Only After Confirmation Using the Junos XML Protocol" on page 294.

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | **288**

Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | **298**

<commit-configuration> | **115**

Commit a Configuration at a Specified Time Using the Junos XML Protocol

In a Junos XML protocol session with a Junos device, a client application can schedule a commit operation at a specified time in the future. The client executes the <commit-configuration> operation and includes the <at-time> element with the specified time.

```
<rpc>
    <commit-configuration>
        <at-time>time</at-time>
    </commit-configuration>
</rpc>
```

To indicate when to perform the commit operation, the client includes the <at-time> element with the time specifier. Table 14 on page 292 outlines the valid types of time specifiers.

Table 14: <at-time> Time Specifiers

Time Specifier	Description	Example
reboot	Commit the configuration the next time the device reboots.	• <at-time>reboot</at-time>
hh:*mm*[:*ss*]	Commit the configuration at the specified time (hours, minutes, and, optionally, seconds). The time must be in the future but before 11:59:59 PM on the current day. Use 24-hour time for the *hh* value. The device interprets the time with respect to its clock and time zone settings.	• Execute the operation at 4:30:00 AM: <at-time>04:30:00</at-time> • Execute the operation at 8:00 PM: <at-time>20:00</at-time>
yyyy-mm-dd hh:*mm*[:*ss*]	Commit the configuration at the specified date and time (year, month, date, hours, minutes, and, optionally, seconds). The specified time must be after you execute the <commit-configuration> operation. Use 24-hour time for the *hh* value. The device interprets the time with respect to its clock and time zone settings.	• Execute the operation at 3:30 PM on August 21, 2005: <at-time>2005-08-21 15:30:00</at-time>

NOTE: The specified time must be more than 1 minute later than the current time on the device.

The Junos XML protocol server immediately checks the configuration for syntactic correctness and returns <rpc-reply>, <commit-results>, and <routing-engine> tag elements. If the syntax check succeeds, the <routing-engine> element encloses the <commit-check-success/>, <name>, and <output> tags. The <name> tag reports the Routing Engine on which the check succeeded, for example, re0 or re1. The <output> element reports the time at which the commit is scheduled.

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>
            <name>(re0 | re1)</name>
            <commit-check-success/>
            <output>commit at will be executed at timestamp</output>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

If the commit check succeeds, the configuration is scheduled for commit at the specified time. The Junos XML protocol server does not emit additional tag elements when it performs the actual commit operation. If the configuration is not syntactically correct, an <xnm:error> element encloses child elements that describe the error. In the case of an error, the commit operation is not scheduled.

The <at-time> tag element can be combined with the <synchronize/> tag, the <log/> tag, or both.

The following example shows how to schedule a commit operation for 10:00 PM on the current day.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <commit-configuration>
    <at-time>22:00</at-time>
  </commit-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <commit-results>
    <routing-engine>
      <name>re1</name>
      <commit-check-success/>
      <output>commit at will be executed at date 22:00:00 timezone</output>
    </routing-engine>
  </commit-results>
</rpc-reply>
```

T1182

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | 288

Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | 298

<commit-configuration> | 115

Commit the Candidate Configuration Only After Confirmation Using the Junos XML Protocol

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 10 minutes, the device automatically loads and commits (rolls back to) the previously committed configuration.

In a Junos XML protocol 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-configuration> and <rpc> tag elements.

```
<rpc>
    <commit-configuration>
        <confirmed/>
    </commit-configuration>
</rpc>
```

To specify a number of minutes for the rollback deadline that is different from the default value of 10 minutes, the application includes the <confirm-timeout> tag element and specifies the number of minutes for the delay, in the range from 1 through 65,535 minutes.

```
<rpc>
    <commit-configuration>
        <confirmed/>
```

```
        <confirm-timeout>rollback-delay</confirm-timeout>
    </commit-configuration>
</rpc>
```

NOTE: You cannot perform a confirmed commit operation on a private copy of the configuration or on an instance of the ephemeral configuration database.

The Junos XML protocol server confirms that it committed the candidate configuration temporarily by returning the <rpc-reply>, <commit-results>, <output>, and <routing-engine> tag elements. If the initial commit operation succeeds, the <routing-engine> element encloses the <commit-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the commit operation succeeded (re0 on devices that use a single Routing Engine, and either re0 or re1 on devices that can have two Routing Engines).

```
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <output>commit confirmed will be automatically rolled back in 10 minutes unless
confirmed</output>
        <routing-engine>
            <name>(re0 | re1)</name>
            <commit-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

If the Junos XML protocol server cannot commit the candidate configuration, the <rpc-reply> element instead encloses an <xnm: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 application emits the <confirmed/> tag in a <commit-configuration> tag element again before the deadline passes. Optionally, it can include 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 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 one of the following tag sequences before the rollback deadline passes:

- The <check/> tag enclosed in <commit-configuration> and <rpc> tag elements. The rollback is canceled and the candidate configuration is committed immediately.

```
<rpc>
    <commit-configuration/>
        <check/>
    </commit-configuration/>
</rpc>
```

- The empty <commit-configuration/> tag enclosed in an <rpc> tag element.

  The rollback is canceled and the candidate configuration is committed immediately, as described in "Committing the Candidate Configuration Using the Junos XML Protocol" on page 288. If the candidate configuration is still the same as the temporarily committed configuration, this effectively recommits the temporarily committed configuration:

```
<rpc>
    <commit-configuration/>
</rpc>
```

- The <synchronize/> tag enclosed in <commit-configuration> and <rpc> tag elements.

```
<rpc>
    <commit-configuration>
        <synchronize/>
    </commit-configuration>
</rpc>
```

  The rollback is canceled and the candidate configuration is checked and committed immediately on both Routing Engines, as described in "Committing and Synchronizing a Configuration on Redundant Control Planes Using the Junos XML Protocol" on page 298. If a confirmed commit operation has been performed on both Routing Engines, then emitting the <synchronize/> tag cancels the rollback on both.

- The <at-time> tag element enclosed in <commit-configuration> and <rpc> tag elements.

```
<rpc>
    <commit-configuration>
        <at-time>time</at-time>
```

```
    </commit-configuration>
</rpc>
```

The rollback is canceled and the configuration is checked immediately for syntactic correctness, then committed at the scheduled time, as described in "Committing a Configuration at a Specified Time Using the Junos XML Protocol" on page 292.

The <confirmed/> and <confirm-timeout> tag elements can be combined with the <synchronize/> tag, the <log/> tag element, or both. For more information, see"Committing and Synchronizing a Configuration on Redundant Control Planes Using the Junos XML Protocol" on page 298 and "Logging a Message About a Commit Operation Using the Junos XML Protocol" on page 305.

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 Junos XML protocol 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 Junos XML Protocol Sessions" on page 98.

The following example shows how to commit the candidate configuration on Routing Engine 1 with a rollback deadline of 20 minutes.

Client Application

```
<rpc>
  <commit-configuration>
    <confirmed/>
    <confirm-timeout>20</confirm-timeout>
  </commit-configuration>
</rpc>
```

Junos XML Protocol Server

```
<rpc-reply xmlns:junos="URL">
  <commit-results>
    <routing-engine>
      <name>re1</name>
      <commit-success/>
    </routing-engine>
  </commit-results>
</rpc-reply>
```

T1152

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | 288

<commit-configuration> | 115

Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol

IN THIS SECTION

A Routing Engine resides within a control plane. For single-chassis configurations, there is one control plane. In redundant systems, there are two control planes, the primary plane and the backup plane. In multichassis configurations, the control plane includes all Routing Engines with the same Routing Engine designation. For example, all primary Routing Engines reside within the *primary* control plane, and all backup Routing Engines reside within the *backup* control plane.

Committing a configuration applies a new configuration to the device Engine. In a multichassis configuration, once a change to the configuration has been committed to the system, this change is propagated throughout the control plane using the distribution function.

In a redundant architecture, you can issue the `synchronize` command to commit the new configuration to both the primary and the backup control planes. When issued, this command saves the current configuration to both device Routing Engines and commits the new configuration to both control planes. On a multichassis system, once the configuration has been committed on both planes, the distribution function distributes the new configuration across both planes. For more information about Routing Engine redundancy, see the Junos OS High Availability User Guide.

NOTE: In a multichassis architecture with redundant control planes, there is a difference between synchronizing the two planes and distributing the configuration throughout each plane. Synchronization only occurs between the Routing Engines within the same chassis. Once this synchronization is complete, the new configuration is distributed to all other Routing Engines within the control planes of other chassis as a separate distribution function.

Because synchronization happens across two separate control planes, synchronizing configurations is only valid on redundant Routing Engine architectures. Further, re0 and re1 configuration groups must be defined on each routing, switching, or security platform. For more information about configuration groups, see the CLI User Guide.

NOTE: If you issue the `synchronize` command on a nonredundant Routing Engine system, the Junos XML protocol server commits the configuration on the one control plane.

For information about synchronizing the ephemeral configuration database, see "Committing and Synchronizing Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol" on page 339. For more information about synchronizing the candidate configuration, see the following sections:

Synchronizing the Candidate Configuration on Both Routing Engines

To synchronize the candidate configuration or private copy on a redundant Routing Engine system, a client application encloses the empty `<synchronize/>` tag in `<commit-configuration>` and `<rpc>` tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
    </commit-configuration>
</rpc>
```

The Junos XML protocol server verifies the configuration's syntactic correctness on the Routing Engine where the `<synchronize/>` tag is emitted (referred to as the local Routing Engine), copies the configuration to the remote Routing Engine and verifies its syntactic correctness there, and then commits the configuration on both Routing Engines.

The Junos XML protocol server encloses its response in `<rpc-reply>` and `<commit-results>` tag elements. It emits a separate `<routing-engine>` tag element for each operation on each Routing Engine:

- If the syntax check succeeds on a Routing Engine, the `<routing-engine>` tag element encloses the `<commit-check-success/>` tag and the `<name>` tag element, which reports the name of the Routing Engine on which the check succeeded (re0 or re1):

  ```
  <routing-engine>
      <name>(re0 | re1)</name>
      <commit-check-success/>
  </routing-engine>
  ```

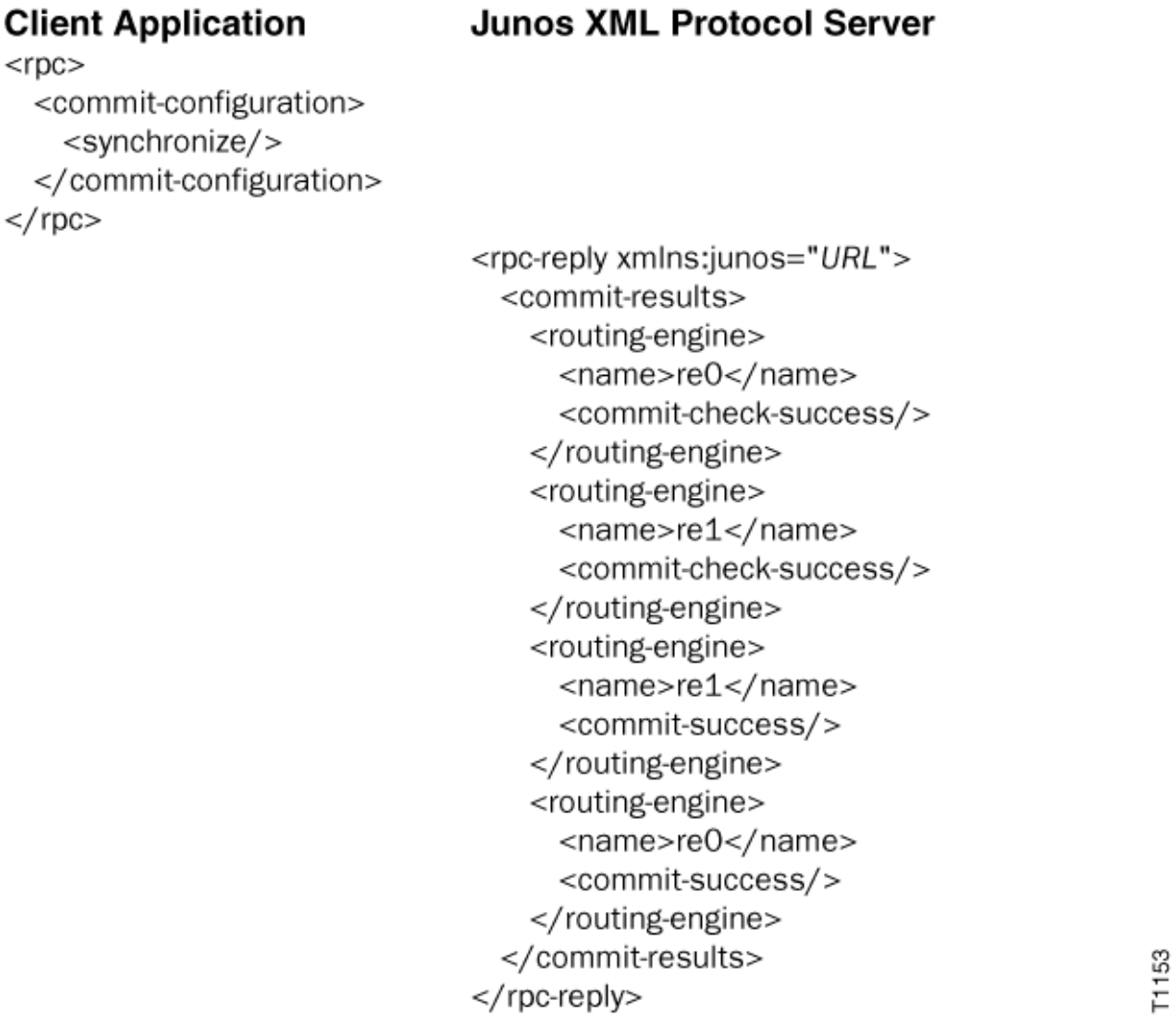
  If the configuration is incorrect, an `<xnm:error>` tag element encloses a description of the error.

- If the commit operation succeeds on a Routing Engine, the <routing-engine> tag element encloses the <commit-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the commit operation succeeded:

```
<routing-engine>
    <name>(re0 | re1)</name>
    <commit-success/>
</routing-engine>
```

  If the commit operation fails, an <xnm:error> tag element encloses a description of the error. The most common causes of failure are semantic or syntactic errors in the configuration.

The following example shows how to commit and synchronize the candidate configuration on both Routing Engines.



Forcing a Synchronized Commit Operation

The synchronize operation fails if the second Routing Engine's candidate configuration is locked. If a synchronization failure occurs, it is best to determine the cause of the failure, take corrective action, and then synchronize the two Routing Engines again. However, when necessary, you can use the <force-synchronize/> command to override a locked configuration and force the synchronization.

NOTE: When you use a force-synchronize command, any uncommitted changes to the configuration will be lost.

To force a synchronization, enclose the empty <synchronize/> and <force-synchronize/> tags in the <rpc> and <commit-configuration> tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <force-synchronize/>
    </commit-configuration>
</rpc>
```

NOTE: In a multichassis environment, synchronization occurs between Routing Engines on the same chassis. Once the synchronization occurs, the configuration changes are propagated across each control plane using the distribution function. If one or more Routing Engines are locked during the distribution of the configuration, the distribution and thus the synchronization will fail. You will need to clear the error in the remote chassis and run the synchronize command again.

The following example shows how to force a synchronization across both Routing Engine planes:

Client Application	Junos XML Protocol Server

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <force-synchronize/>
    </commit-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos=
             "http://xml.juniper.net/junos/9.010/junos">
    <commit-results>
       <routing-engine junos:style="show-name">
            <name>re0</name>
            <commit-check-success/>
        </routing-engine>
       <routing-engine junos:style="show-name">
            <name>re1</name>
            <commit-success/>
        </routing-engine>
       <routing-engine junos:style="show-name">
            <name>re0</name>
            <commit-success/>
        </routing-engine>
    </commit-resuls>
</rpc-reply>
```

Synchronizing the Candidate Configuration Simultaneously with Other Operations

The <synchronize/> tag can be combined with the other tag elements that can occur within the <commit-configuration> tag element. The Junos XML protocol server checks, copies, and commits the configuration, and emits the same response tag elements as when the <synchronize/> tag is used by itself. The possible combinations are described in the following sections.

Verifying the Configuration on Both Routing Engines

To check the syntactic correctness of a local configuration on both Routing Engines without committing it, the application encloses the <synchronize/> and <check/> tag elements in <commit-configuration> and <rpc> tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <check/>
    </commit-configuration>
</rpc>
```

The <force-synchronize/> tag cannot be combined with the <check/> tag elements.

For more information about verifying configurations, see "Verifying Configuration Syntax Using the Junos XML Protocol" on page 287.

Scheduling Synchronization for a Specified Time

To commit a configuration on both Routing Engines at a specified time in the future, the application encloses the <synchronize/> and <at-time> tag elements in <commit-configuration> and <rpc> tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <at-time>time</at-time>
    </commit-configuration>
</rpc>

<rpc>
    <commit-configuration>
        <force-synchronize/>
        <at-time>time</at-time>
    </commit-configuration>
</rpc>
```

As when the <at-time> tag element is emitted by itself, the Junos XML protocol server verifies syntactic correctness immediately and does not emit additional tag elements when it actually performs the commit operation on each Routing Engine.

Synchronizing Configurations but Requiring Confirmation

To commit the candidate configuration on both Routing Engines but require confirmation for the commit to become permanent, the application encloses the <synchronize/>, <confirmed/>, and (optionally) <confirm-timeout> tag elements in <commit-configuration> and <rpc> tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <confirmed/>
        [<confirm-timeout>minutes</confirm-timeout>]
    </commit-configuration>
</rpc>
```

The same rollback deadline applies to both Routing Engines and can be extended on both at once by again emitting the <synchronize/>, <confirmed/>, and (optionally) <confirm-timeout> tag elements on the Routing Engine where the tag elements were emitted the first time.

The <force-synchronize/> tag cannot be combined with the <confirmed/> and <confirm-timeout> tag elements.

For more information about confirmed commit operations, see "Committing the Candidate Configuration Only After Confirmation Using the Junos XML Protocol" on page 294.

Logging a Message About Synchronized Configurations

To synchronize configurations and record a log message when the commit succeeds on each Routing Engine, the application encloses the <synchronize/> and <log/> tag elements in <commit-configuration> and <rpc> tag elements:

```
<rpc>
    <commit-configuration>
        <synchronize/>
        <log>message</log>
    </commit-configuration>
</rpc>

<rpc>
    <commit-configuration>
        <force-synchronize/>
        <log>message</log>
    </commit-configuration>
</rpc>
```

The commit operation proceeds as previously described in the <synchronize/> or <force-synchronize/> tag descriptions. The message for each Routing Engine is recorded in the commit history log maintained by that Routing Engine. For more information about logging, see "Logging a Message About a Commit Operation Using the Junos XML Protocol" on page 305.

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | 288

Log a Message About a Commit Operation Using the Junos XML Protocol | 305

<commit-configuration> | 115

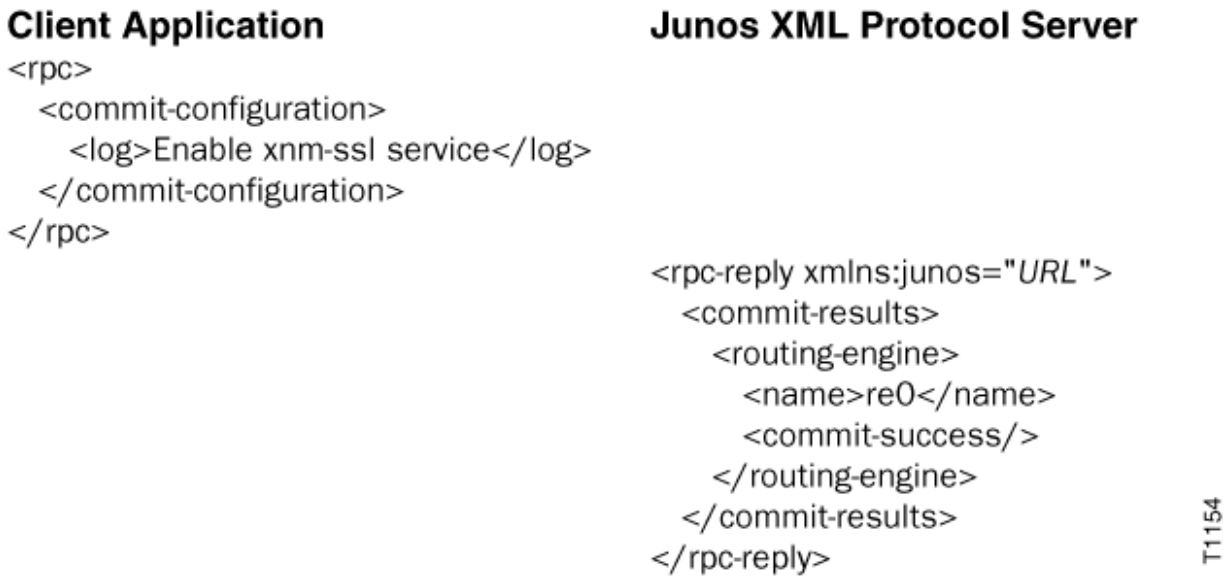
Log a Message About a Commit Operation Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, to log a comment when performing a commit operation, a client application encloses the <log> tag element in <commit-configuration> and <rpc> tag elements:

```
<rpc>
    <commit-configuration>
        <log>message</log>
    </commit-configuration>
</rpc>
```

The <log> element can be combined with other tags within the <commit-configuration> tag element (the <at-time>, <confirmed/>, and <confirm-timeout>, or <synchronize/> tag elements) and does not change the effect of the operation. When the <log> tag element is emitted by itself, the associated commit operation begins immediately.

The following example shows how to log a message as the candidate configuration is committed.



The commit history, which includes any commit comments, stores an entry for each pending commit and up to 50 previous commits for the standard configuration database. To request the history, a client

application encloses the <get-commit-information/> tag in <rpc> tag elements. The equivalent operational mode CLI command is show system commit.

```
<rpc>
    <get-commit-information/>
</rpc>
```

The Junos XML protocol server encloses the information in <commit-information> and <rpc-reply> tag elements. For information about the child tag elements of the <commit-information> tag element, see its entry in the *Junos XML API Operational Developer Reference*.

```
<rpc-reply xmlns:junos="URL">
    <commit-information>
        <!-- tag elements representing the commit log -->
    </commit-information>
</rpc-reply>
```

The following example shows how to request the commit log.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-commit-information/>
</rpc>
                <rpc-reply xmlns:junos="URL">
                  <commit-information>
                    <commit-history>
                      <sequence-number>0</sequence-number>
                      <user>barbara</user>
                      <client>other</client>
                      <date-time junos:seconds="1058370173">2003-07-16 08:42:53 PDT</date-time>
                      <log>Enable xnm-ssl service</log>
                    </commit-history>
                    <commit-history>
                      <sequence-number>1</sequence-number>
                      <user>root</user>
                      <client>other</client>
                      <date-time junos:seconds="1058322166">2003-07-15 19:22:46 PDT</date-time>
                    </commit-history>
                    <commit-history>
                      <sequence-number>2</sequence-number>
                      <user>root</user>
                      <client>cli</client>
                      <date-time junos:seconds="1058219717">2003-07-14 14:55:17 PDT</date-time>
                    </commit-history>
                            .
                            .
                            .
                  </commit-information>
                </rpc-reply>
```

T1183

RELATED DOCUMENTATION

- Commit the Candidate Configuration Using the Junos XML Protocol | 288
- Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | 298
- <commit-configuration> | 115

View the Configuration Revision Identifier for Determining Synchronization Status of Devices with NMS

The configuration revision identifier (CRI) is a unique string that is associated with a committed configuration. Network management system (NMS) applications, such as Junos Space, can use the CRI to detect if other systems made out-of-band (OOB) configuration changes to the network device. Out-

of-band configuration changes are configuration changes made to a device outside of the NMS. For example, you can perform configuration changes on a device using the CLI, the J-Web interface, or the Junos Space Network Management Platform configuration editor.

The NMS application can cache the CRI for a given commit. At a later date, the NMS can compare the cached value to the CRI of the current configuration on the network device. By comparing the values for differences, the NMS can detect if other systems made OOB configuration changes to the device. Monitoring the CRI might not be necessary if the NMS application is the only utility that modifies the device configuration. However, in a real-world network deployment, OOB configuration commits might occur on a device, such as during a maintenance window for support operations. In such cases, the NMS application might not detect these OOB commits.

After a successful commit, the `<commit-results>` tag includes a `<commit-revision-information>` tag. The `<commit-revision-information>` tag includes the previous revision number and updated revision number. The NMS application can store this revision number locally. At a later time, the NMS application can retrieve the latest revision number from the network device and compare it against the revision number stored locally to validate whether it is out-of-sync with the device.

The following example RPC reply includes the `<commit-revision-information>` tag containing the commit revision details:

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/23.2R1/junos">
  <commit-results>
    <routing-engine junos:style="normal">
      <name>re0</name>
      <commit-check-success/>
      <commit-success/>
      <commit-revision-information>
        <new-db-revision>re0-1760600165-115</new-db-revision>
        <old-db-revision>re0-1760586277-114</old-db-revision>
      </commit-revision-information>
    </routing-engine>
</commit-results>
</rpc-reply>
```

The configuration revision identifier is a string, which has the following format:

```
<routing-engine-name>-<timestamp>-<counter>
```

The NMS application considers the CRI as an entire string and does not parse individual substrings of this identifier. The Routing Engine name is different from the user-configured hostname of the device. The Routing Engine name identifies the source of the configuration change. Different platforms use

different Routing Engine names. The revision number's timestamp uses the Unix epoch format (also known as Unix time or POSIX time). The counter index increments by one for every commit operation on the device.

Starting in Junos OS Release 20.4R1 and Junos OS Evolved Release 20.4R1, an application can also use the CRI associated with a committed configuration to:

- View the configuration.
- Compare two configurations.
- Revert to the configuration.
- Retrieve the current rollback index associated with that configuration.

RELATED DOCUMENTATION

Commit the Candidate Configuration Using the Junos XML Protocol | **288**

Commit and Synchronize a Configuration on Redundant Control Planes Using the Junos XML Protocol | **298**

<commit-revision-information> | **155**

<commit-configuration> | **115**

CHAPTER 13

Ephemeral Configuration Database

IN THIS CHAPTER

Understanding the Ephemeral Configuration Database

IN THIS SECTION

The *ephemeral database* is an alternate configuration database that provides a fast programmatic interface for performing configuration updates on devices running Junos OS and Junos OS Evolved. The ephemeral database enables Juniper Extension Toolkit (JET) applications and NETCONF and Junos XML management protocol client applications to concurrently load and commit configuration changes to a device and with significantly greater throughput than when committing data to the candidate configuration database.

Use Feature Explorer to confirm platform and release support for specific features.

The following sections discuss the different aspects of the ephemeral configuration database.

Benefits of the Ephemeral Configuration Database

- Enables multiple client applications to concurrently configure a device by loading and committing data to separate instances of the ephemeral database
- Enables fast provisioning and rapid configuration changes in dynamic environments that require fast commit times

Ephemeral Configuration Database Overview

When managing Junos devices, the recommended and most common method to configure the device is to modify and commit the candidate configuration, which corresponds to a persistent (static) configuration database. The standard commit operation handles configuration groups, macros, and commit scripts; performs commit checks to validate the configuration's syntax and semantics; and stores copies of the committed configurations. The standard commit model is robust because it prevents configuration errors and it enables you to roll back to a previously committed configuration. However, in some cases, the commit operation can consume a significant amount of time and device resources.

JET applications and NETCONF and Junos XML protocol client applications can also configure the ephemeral database. The ephemeral database is an alternate configuration database that provides a configuration layer separate from both the candidate configuration database and the configuration layers of other client applications. The ephemeral commit model enables Junos devices to commit and merge changes from multiple clients and execute the commits with significantly greater throughput than when committing data to the candidate configuration database. Thus, the ephemeral database is advantageous in dynamic environments where fast provisioning and rapid configuration changes are required, such as in large data centers.

A commit operation on the ephemeral database requires less time than the same operation on the static database because the ephemeral database is not subject to the same validation required in the static database. As a result, the ephemeral commit model provides better performance than the standard commit model but at the expense of some of the more robust features present in the standard model. The ephemeral commit model has the following restrictions:

- Configuration data syntax is validated, but configuration data semantics are not validated.
- Certain configuration statements are not supported as described in "Unsupported Configuration Statements in the Ephemeral Configuration Database" on page 324.
- Configuration groups and interface ranges are not processed.
- Macros, commit scripts, and translation scripts are not processed.
- Previous versions of the ephemeral configuration are not archived.

- Standard show commands do not display ephemeral configuration data in the output.
- Ephemeral configuration data does not persist when you:
  - Install a package that requires rebuilding the Junos schema, for example, an OpenConfig or YANG package.
  - Perform a software upgrade or a unified in-service software upgrade (ISSU).
  - Reboot or power cycle the device.

CAUTION: We strongly recommend that you exercise caution when using the ephemeral configuration database. Committing invalid configuration data can corrupt the ephemeral database, which can cause Junos processes to restart or stop responding and result in disruption to the system or network.

Junos devices validate the syntax but not the semantics, or constraints, of the configuration data committed to the ephemeral database. For example, if the configuration references an undefined routing policy, the configuration might be syntactically correct, but it would be semantically incorrect. The standard commit model generates a commit error in this case, but the ephemeral commit model does not. Therefore, it is imperative to validate all configuration data before committing it to the ephemeral database. If you commit configuration data that is invalid or results in undesirable network disruption, you must remove the problematic data from the database. You can delete the data, or if necessary, you can reboot the device, which deletes the configuration data in all instances of the ephemeral configuration database.

NOTE: The ephemeral configuration database stores internal version information in addition to configuration data. As a result, the size of the ephemeral configuration database is always larger than the static configuration database for the same configuration data, and most operations on the ephemeral database, whether additions, modifications, or deletions, increase the size of the database.

NOTE: When you use the ephemeral configuration database, commit operations on the static configuration database might take longer, because the device must perform additional operations to merge the static and ephemeral configuration data.

Ephemeral Database Instances

Junos devices provide a default ephemeral database instance, which is automatically enabled. You can also enable multiple user-defined instances of the ephemeral configuration database. JET applications and NETCONF and Junos XML protocol client applications can concurrently load and commit data to

separate instances of the ephemeral database. The active device configuration is a merged view of the static and ephemeral configuration databases.

Ephemeral database instances are useful when multiple client applications need to simultaneously update a device configuration. For example, two or more SDN controllers might need to simultaneously push configuration data to the same device. In the standard commit model, one controller might have an exclusive lock on the candidate configuration, thereby preventing the other controller from modifying it. By using separate ephemeral instances, the controllers can deploy the changes at the same time.

NOTE: Applications can simultaneously load and commit data to different ephemeral database instances in addition to the static configuration database. However, the device processes the commits sequentially. As a result, the commit to a specific database might be delayed, depending on the processing order.

The Junos processes read the configuration data from both the static configuration database and the ephemeral configuration database. When one or more ephemeral database instances are in use and there is conflicting data, statements in a database with a higher priority override the statements in a database with a lower priority. The database priority, from highest to lowest, is as follows:

1. Statements in a user-defined instance of the ephemeral configuration database.

   If the device uses multiple user-defined ephemeral instances, it determines the priority by the order in which the instances are configured at the `[edit system configuration-database ephemeral]` hierarchy level, running from highest to lowest priority.

2. Statements in the default ephemeral database instance.

3. Statements in the static configuration database.

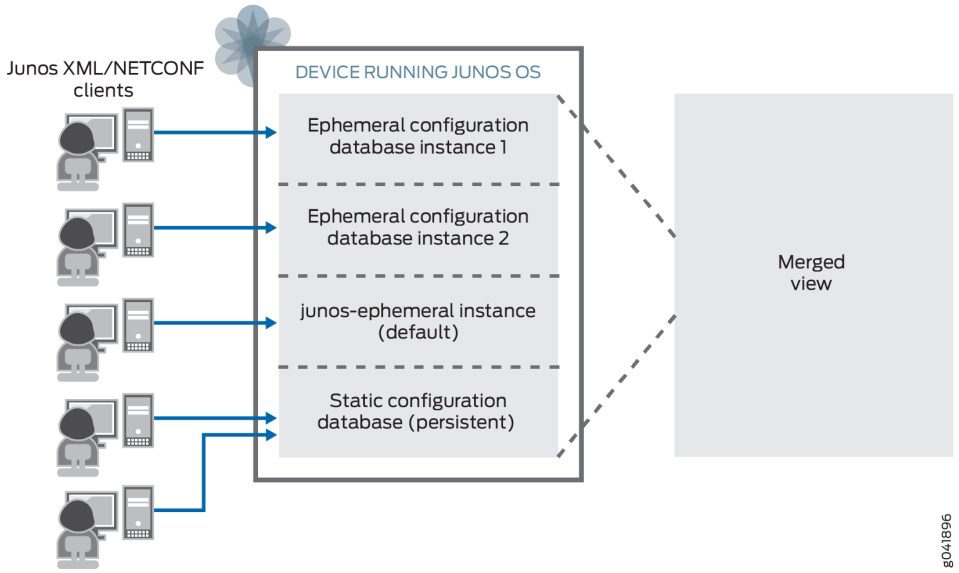
Consider the following configuration:

```
system {
    configuration-database {
        ephemeral {
            instance 1;
            instance 2;
        }
    }
}
```

Figure 1 on page 314 illustrates the priority of the ephemeral database instances and the static (committed) configuration database. In this example, ephemeral database instance 1 has the highest

priority, followed by ephemeral database instance 2, then the default ephemeral database instance, and finally the static configuration database.

Figure 1: Ephemeral Database Instances



Ephemeral Database General Commit Model

JET applications and NETCONF and Junos XML protocol client applications can modify the ephemeral configuration database. JET applications must send configuration requests as pairs of load and commit operations. NETCONF and Junos XML protocol client applications can perform multiple load operations before executing a commit operation.

CAUTION: You must validate all configuration data before loading it into the ephemeral database and committing it on the device. Committing invalid configuration data can cause Junos processes to restart or stop responding and result in disruption to the system or network.

Client applications can simultaneously load and commit data to different ephemeral database instances. Commits issued at the same time for different ephemeral instances are queued and processed serially by the device. If a client disconnects from a session, the device discards any uncommitted configuration changes in the ephemeral instance. However, configuration data that has already been committed to the ephemeral instance by that client is unaffected.

When you commit an ephemeral instance, the system validates the syntax, but not the semantics, of the ephemeral configuration data. When the commit is complete, the device notifies the affected system processes. The processes read the updated configuration and merge the ephemeral data into the active configuration according to the rules of prioritization described in "Ephemeral Database Instances" on page 312. The active device configuration is a merged view of the static and ephemeral configuration databases.

NOTE: The ephemeral database's commit time will be slightly longer on devices running Junos OS Evolved than on devices running Junos OS because of the architectural differences between the two operating systems.

For detailed information about committing and synchronizing instances of the ephemeral configuration database, see "Commit and Synchronize Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol" on page 339.

Using the Ephemeral Database with High Availability Features

High availability refers to the hardware and software components that provide redundancy and reliability for network communications. You should consider certain behaviors and caveats before using the ephemeral database on systems that use high availability features. High availability features include redundant Routing Engines, graceful Routing Engine switchover (GRES), nonstop active routing (NSR), and interchassis redundancy for MX Series routers or EX Series switches using Virtual Chassis. The following sections describe these behaviors and outline how the different ephemeral database commit synchronize models can affect these behaviors.

Understanding Ephemeral Database Commit Synchronize Models

The ephemeral configuration database has two models for synchronizing ephemeral configuration data across Routing Engines or Virtual Chassis members during a commit synchronize operation:

- Asynchronous
- Synchronous

Starting with Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the ephemeral database uses the synchronous model by default and devices that enable GRES or NSR must use the synchronous model. However, you can still use either model to synchronize ephemeral data on devices that do not enable GRES or NSR. In earlier releases, the asynchronous model is the default.

Asynchronous Model

When the ephemeral database uses the asynchronous commit model, the primary Routing Engine or Virtual Chassis primary device commits the configuration and then notifies the backup device. The

requesting Routing Engine does not wait for the other Routing Engine to first synchronize and commit the configuration. Devices that use high availability features require that the primary and backup Routing Engines are synchronized in case of a failover. However, there can be situations in which an asynchronous commit synchronize operation can be interrupted and fail to synchronize the ephemeral configuration to the other Routing Engine.

Synchronous Model

The synchronous commit model is similar to the model used by the static configuration database. Synchronous commit operations are slower than asynchronous commit operations, but they provide better assurance that the ephemeral configuration is synchronized across Routing Engines or Virtual Chassis members. Thus, the synchronous commit model enables you to use the ephemeral database with greater reliability on devices that use high availability features.

In a dual Routing-Engine or MX Series Virtual Chassis environment, the synchronous commit model works as follows:

1. The primary Routing Engine or Virtual Chassis primary device starts its commit operation for the ephemeral instance.
2. At a given point during the commit operation, the device initiates a commit on the backup Routing Engine or Virtual Chassis backup device.
3. If the other Routing Engine successfully commits the configuration, then the primary Routing Engine continues its commit operation. If the commit fails on the other Routing Engine, then the primary Routing Engine also fails the commit.

When an EX Series Virtual Chassis uses the synchronous commit model, the member switch in the primary Routing Engine role first initiates the commit operation on the other members simultaneously. Because an EX Series Virtual Chassis can have many members, the primary switch then proceeds with its commit operation, even if the commit fails on another member.

NOTE: As is the case for the static configuration database, even with the synchronous commit synchronize model, there can be rare circumstances in which the device commits an updated ephemeral configuration on the backup Routing Engine but fails to complete the commit on the primary Routing Engine resulting in the configurations being out of synchronization.

Failover Synchronization

Devices running Junos OS Release 20.2R1 or later and devices running Junos OS Evolved also support failover configuration synchronization for the ephemeral database. If you configure failover synchronization, then when the backup Routing Engine synchronizes with the primary Routing Engine, for example, when it is newly inserted, brought back online, or during a change in role, it synchronizes

both its static and ephemeral configuration databases. To enable failover synchronization, configure the `commit synchronize` statement at the `[edit system]` hierarchy level in the static configuration database.

NOTE: For failover synchronization, the backup Routing Engine or the MX Virtual Chassis backup device only synchronizes the ephemeral configuration database with the primary device if both the backup device and the primary device are running the same software version.

Both commit synchronize operations and failover synchronize operations synchronize the ephemeral configuration data to the other Routing Engine using a load update operation instead of a load override operation. By using a load update operation, the device only needs to notify the Junos processes that correspond to changed statements during the update, which minimizes possible disruptions to the network.

Redundant Routing Engines

Dual Routing Engine systems support configuring the ephemeral database. However, the ephemeral commit model does not automatically synchronize ephemeral configuration data to the backup Routing Engine during a commit operation. You can synchronize the data in an ephemeral instance on a per-commit or per-session basis. You can also configure an ephemeral instance to automatically synchronize its data every time you commit the instance. For more information, see "Commit and Synchronize Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol" on page 339.

NOTE: Multichassis environments do not support synchronizing the ephemeral configuration database to the other Routing Engines.

When a client application commits data in an ephemeral instance and synchronizes it to the backup Routing Engine, the device synchronizes the ephemeral data using the configured commit synchronize model. Dual Routing Engine devices also support failover configuration synchronization for the ephemeral database. For more information, see "Understanding Ephemeral Database Commit Synchronize Models" on page 315.

Graceful Routing Engine Switchover (GRES)

GRES enables a device with redundant Routing Engines to continue forwarding packets, even if the primary Routing Engine fails. GRES requires that the primary and backup Routing Engines synchronize the configuration and certain state information before a switchover occurs.

We recommend that you use the synchronous commit synchronize model on devices that enable GRES. Moreover, starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES must use the synchronous model. Synchronous commit operations are slower than

asynchronous commit operations, but they provide better assurance that the ephemeral configuration is synchronized between Routing Engines.

NOTE: Dual Routing Engine devices running Junos OS Evolved enable GRES by default.

Although supported in certain releases, we do *not* recommend using the asynchronous commit synchronize model on devices that enable GRES. If you use the asynchronous model, in certain circumstances, the ephemeral database might not be synchronized between the primary and backup Routing Engines when a switchover occurs. For example, the backup and primary Routing Engines might not synchronize the ephemeral database if the commit synchronize operation is interrupted by a sudden power outage. If you use the asynchronous commit model on a GRES-enabled device, you must explicitly configure the device to synchronize ephemeral configuration data to the backup Routing Engine. To enable synchronization, configure the `allow-commit-synchronize-with-gres` statement at the `[edit system configuration-database ephemeral]` hierarchy level in the static configuration database.

Nonstop Active Routing (NSR)

Nonstop active routing (NSR) enables the transparent switchover of the Routing Engines in the event that one of the Routing Engines goes down. We recommend that you use the synchronous commit synchronize model on devices that enable NSR. Moreover, starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable NSR must use the synchronous model. Synchronous commit operations are slower than asynchronous commit operations, but they provide better assurance that the ephemeral configuration is synchronized between Routing Engines.

Although supported in certain releases, we do *not* recommend using the asynchronous commit synchronize model on devices that enable NSR, because it comes with certain caveats. In a deployment with dual Routing Engines, a commit synchronize operation on an ephemeral instance on the primary Routing Engine results in an asynchronous commit on the backup Routing Engine. If the device notifies the routing protocol process (rpd) in the process of updating the configuration, it could result in an undesirable behavior of the system due to the asynchronous nature of the commit on the backup Routing Engine. In Junos OS Release 21.1R1 and later, the device synchronizes the ephemeral instance to the backup Routing Engine using a load update operation, so it only notifies processes corresponding to statements that are changed.

NOTE: Applications utilizing the ephemeral database are only impacted in this NSR situation if they interact with the routing protocol process. For example, the SmartWall Threat Defense Director (SmartWall TDD) would not be impacted in this case, because it only interacts with the firewall process (dfwd) through the ephemeral database.

MX Series Virtual Chassis

MX Series Virtual Chassis support configuring the ephemeral database. You can configure and commit an ephemeral instance only on the primary Routing Engine of the Virtual Chassis primary device.

An MX Series Virtual Chassis does not automatically synchronize any ephemeral configuration data during a commit operation. As with dual Routing Engine systems, you can synchronize the data in an ephemeral instance on a per-commit or per-session basis. You can also configure an ephemeral instance to automatically synchronize its data every time you commit the instance. The device synchronizes the ephemeral data only from the primary Routing Engine on the primary device to the primary Routing Engine on the backup device.

NOTE: MX Series Virtual Chassis do not, under any circumstance, synchronize ephemeral configuration data from the primary Routing Engine to the backup Routing Engine on the respective Virtual Chassis member.

MX Series Virtual Chassis must have GRES configured. If you configure the ephemeral database to use the synchronous commit synchronize model (recommended), the device synchronizes the ephemeral instance to the other Routing Engine when you request a commit synchronize operation. However, if the ephemeral database uses the asynchronous commit synchronize model, you must explicitly configure the `allow-commit-synchronize-with-gres` statement in the static configuration database to enable synchronization. See "Understanding Ephemeral Database Commit Synchronize Models" on page 315 for more information about the ephemeral database commit models.

NOTE: Starting with Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES must use the synchronous commit synchronize model.

When you commit and synchronize an ephemeral instance on an MX Series Virtual Chassis that uses the asynchronous commit synchronize model:

1. The Virtual Chassis primary device validates the configuration syntax and commits the ephemeral instance on its primary Routing Engine.
2. If the commit is successful, the primary device notifies the backup device to synchronize the ephemeral instance.
3. The backup device commits the ephemeral instance on its primary Routing Engine only. If the commit operation fails, the backup device logs a message in the system log file but does not notify the primary device.

When you commit and synchronize an ephemeral instance on an MX Series Virtual Chassis that is configured to use the synchronous commit synchronize model, which is the recommended method:

1. The Virtual Chassis primary device starts its commit of the ephemeral instance on its primary Routing Engine.
2. At a given point in its commit operation, the primary device initiates a commit on the backup device's primary Routing Engine.
3. If the backup device successfully commits the configuration, then the primary device proceeds with its commit operation. If the backup device fails to commit the configuration, then the primary device also fails the commit.

As outlined, when you use the asynchronous commit synchronize model for the ephemeral database, the commit can succeed on the primary device but fail on the backup device. When you use the synchronous commit synchronize model, the commit either succeeds or fails for both primary Routing Engines, except in rare circumstances.

MX Series Virtual Chassis support failover configuration synchronization for the ephemeral database. To configure failover configuration synchronization, include the `commit synchronize` statement at the `[edit system]` hierarchy level in the static configuration database. After you configure the statement, the primary Routing Engine on the Virtual Chassis backup device synchronizes both its static and ephemeral configuration databases when it synchronizes with the primary Routing Engine on the Virtual Chassis primary device, for example, after it restarts.

NOTE: For failover synchronization, the MX Virtual Chassis backup device only synchronizes the ephemeral configuration database with the primary device if both devices are running the same software version.

EX Series Virtual Chassis

EX Series Virtual Chassis support the ephemeral configuration database. You can only configure and commit an ephemeral instance on the member switch in the primary Routing Engine role. Starting in Junos OS Release 23.4R1, you can synchronize the ephemeral database across EX Series Virtual Chassis members.

An EX Series Virtual Chassis does not automatically synchronize any ephemeral configuration data during a commit operation. You can synchronize the data in an ephemeral instance on a per-commit or per-session basis. You can also configure an ephemeral instance to automatically synchronize its data every time you commit the instance.

You can configure GRES on an EX Series Virtual Chassis to enable the Virtual Chassis to continue forwarding packets if the primary Routing Engine fails. If you configure the ephemeral database to use the synchronous commit synchronize model (recommended), the device synchronizes the ephemeral instance to the other members when you request a commit synchronize operation. However, if the ephemeral database uses the asynchronous commit synchronize model and GRES is configured, you

must explicitly configure the `allow-commit-synchronize-with-gres` statement in the static configuration database to enable synchronization.

NOTE: Starting with Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES must use the synchronous commit synchronize model.

When you commit and synchronize an ephemeral instance on an EX Series Virtual Chassis that uses the asynchronous commit synchronize model:

1. The member switch in the primary Routing Engine role validates the configuration syntax and commits the ephemeral instance.
2. If the commit is successful, the primary device notifies the `commit-syncd` process, which initiates the commit on each member switch in turn.
3. Each member switch commits the ephemeral instance. If the commit operation fails on any member, it does not affect the commit operation on the other members.

When you commit and synchronize an ephemeral instance on an EX Series Virtual Chassis that is configured to use the synchronous commit synchronize model, which is the recommended method:

1. The member switch in the primary Routing Engine role initiates the commit on all member switches simultaneously.
2. Each member switch commits the ephemeral instance and notifies the primary switch. If the commit operation fails on any member, it does not affect the commit operation on the other members.
3. After receiving responses from all member switches, the primary switch commits the ephemeral instance.

As outlined, in the asynchronous model, the primary switch relies on the `commit-syncd` process to initiate the commits on each member switch sequentially. If the `commit-syncd` process fails for any reason, then some commits might not be initiated. In the synchronous commit model, the primary switch initiates the commit on all member switches directly and in parallel. Thus, the synchronous commit model is generally more reliable than the asynchronous commit model. In either case, if the commit fails on one member, it does not impact or prevent the commit on the other members.

Additionally, in the synchronous commit model, the primary switch displays the commit progress for each member as the commit occurs. In the asynchronous model, the commits occur in the background, so in this case, only the primary device logs the commit results.

Ephemeral Database Best Practices

The ephemeral configuration database enables multiple applications to make rapid configuration changes simultaneously. Because the ephemeral configuration database does not use the same

safeguards as the static configuration database, you should carefully consider how you use the ephemeral database. We recommend following these best practices to optimize performance and avoid potential issues when you use the ephemeral configuration database.

Regulate Commit Frequency

The ephemeral database is designed for faster commits. However, committing too frequently can cause problems if the applications that consume the configuration can't keep pace with the rate of commit operations. Therefore, we recommend that you commit the next set of changes only after the device's operational state reflects the changes from the previous commit.

For example, if you execute frequent, rapid commits, the device could overwrite certain configuration data that it stores in external files before a Junos process reads the previous update. If a Junos process misses an important update, the device or network could exhibit unpredictable behavior.

Ensure Data Integrity

Junos devices do not validate configuration data semantics when you commit data to an ephemeral database. Therefore, you must take additional steps before loading and committing the configuration to ensure data integrity. We recommend that you always:

- Validate configuration data before loading it in the database
- Consolidate related configuration statements into a single database

You should validate all configuration data before loading it into an ephemeral database. We recommend that you pre-validate your configuration data using a static database, which validates both syntax and semantics.

Additionally, you should always load related configuration data into a single database. Adding related or dependent configuration data in the same database helps ensure that the device can detect and process related statements during a commit operation. For example, if you define a firewall filter in the static configuration database or in an ephemeral configuration database, then you should configure the application of the filter to an interface in the same configuration database.

By contrast, suppose you configure some statements in the static database but you configure related or dependent statements in an ephemeral database. When you commit the static database, the system validates the data only within that database. The system might not identify the dependent configuration in the ephemeral database, which can cause the validation, and thus the commit, to fail.

Consolidate Scaled Configurations

We recommend that you load scaled configurations into a single ephemeral database instance, rather than distributing them across multiple databases. A scaled configuration might include, for example, large lists of:

- Policy options
- Prefix lists
- VLANs
- Firewall filters

When you restrict a top-level configuration hierarchy to a single database, internal optimizations enable Junos processes to consume the configuration more efficiently. Alternatively, if you spread the configuration across multiple databases, Junos processes must parse a merged view of the configuration, which generally requires more resources and processing time.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the default commit synchronize model is `synchronous`. In earlier releases, the default is `asynchronous`.
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES or NSR must use the synchronous commit synchronize model.
21.1R1	Starting in Junos OS Release 21.1R1, the device synchronizes the ephemeral database instance to the backup Routing Engine using a load update operation instead of a load override operation. As a result, Junos OS only notifies processes corresponding to the changed statements.
20.2R1	Starting in Junos OS Release 20.2R1, when you configure the `commit synchronize` statement at the `[edit system]` hierarchy level in the static configuration database and the backup Routing Engine synchronizes with the primary Routing Engine, for example, when it is newly inserted, brought back online, or during a change in role, it synchronizes both its static and ephemeral configuration databases. In earlier releases, the device synchronizes only the static configuration database.
20.2R1	Starting in Junos OS Release 20.2R1, if an ephemeral database instance is already synchronized between the primary and backup Routing Engines, and you update the ephemeral instance on the primary Routing Engine, Junos OS only notifies the processes corresponding to the changed configuration when it commits the changes on the backup Routing Engine.

18.2R1	Starting in Junos OS Release 18.2R1, devices running Junos OS support configuring up to seven user-defined instances of the ephemeral configuration database. In earlier releases, you can configure up to eight user-defined instances.

RELATED DOCUMENTATION

Enable and Configure Instances of the Ephemeral Configuration Database | 328

Example: Configuring the Ephemeral Configuration Database Using NETCONF

Unsupported Configuration Statements in the Ephemeral Configuration Database

IN THIS SECTION

The ephemeral database is an alternate configuration database. Juniper Extension Toolkit (JET) applications and NETCONF and Junos XML protocol client applications can simultaneously load and commit configuration changes to the ephemeral database with significantly greater throughput than when committing data to the candidate configuration database. To improve commit performance, the ephemeral commit process does not perform all of the operations and validations executed by the standard commit model. As a result, you cannot configure some features through the ephemeral database. For example, the ephemeral configuration database does not support configuring interface alias names.

Use Feature Explorer to confirm platform and release support for specific features.

Review the "Platform-Specific Ephemeral Database Behavior" on page 327 section for notes related to your platform.

The ephemeral configuration database does *not* support the following configuration statements. We've grouped the statements under their top-level configuration statement hierarchy. If a client attempts to configure an unsupported statement in an ephemeral instance, the server returns an error during the load operation.

[edit]

```
[edit apply-groups]
[edit access]
[edit chassis]
[edit dynamic-profiles]
[edit security] (SRX Series only)
```

[edit interfaces]

```
[edit interfaces interface-name unit logical-unit-number alias alias-name]
[edit interfaces interface-range]
```

[edit logical-systems]

```
[edit logical-systems logical-system-name interfaces interface-name unit logical-unit-number
alias alias-name]
[edit logical-systems logical-system-name policy-options prefix-list name apply-path path]
[edit logical-systems logical-system-name system processes routing]
```

NOTE: Starting in Junos OS Release 23.2R1 and Junos OS Evolved Release 23.4R1, you can configure MSTP, RSTP, and VSTP in the ephemeral configuration database on supported platforms.

[edit policy-options]

```
[edit policy-options prefix-list name apply-path path]
```

[edit protocols]

```
[edit protocols igmp]
[edit protocols mld]
```

NOTE: Starting in Junos OS Release 23.2R1 and Junos OS Evolved Release 23.4R1, you can configure MSTP, RSTP, and VSTP in the ephemeral configuration database on supported platforms.

[edit routing-instances]

> **NOTE**: Starting in Junos OS Release 23.2R1 and Junos OS Evolved Release 23.4R1, you can configure MSTP, RSTP, and VSTP in the ephemeral configuration database on supported platforms.

[edit security]

```
[edit security group-vpn member ipsec vpn]
[edit security ssh-known-hosts host hostname]
```

[edit services]

```
[edit services ssl initiation profile]
[edit services ssl proxy profile]
[edit services ssl termination profile]
```

[edit system]

```
[edit system archival]
[edit system commit delta-export]
[edit system commit fast-synchronize]
[edit system commit notification]
[edit system commit peers]
[edit system commit peers-synchronize]
[edit system commit persist-groups-inheritance]
[edit system commit server]
[edit system compress-configuration-files]
[edit system configuration-database]
[edit system extensions]
[edit system fips]
[edit system host-name]
[edit system license]
[edit system login]
[edit system master-password]
[edit system max-configurations-on-flash]
[edit system radius-options]
[edit system regex-additive-logic]
[edit system scripts]
```

```
[edit system services extension-service notification allow-clients address]
[edit system time-zone]
```

Platform-Specific Ephemeral Database Behavior

Use Feature Explorer to confirm platform and release support for specific features.

Use the following table to review platform-specific behaviors for your platforms.

Table 15: Platform-Specific Behavior

Platform	Difference
SRX Series	• On SRX Series devices that support the ephemeral database, you cannot configure the `[edit security]` statement hierarchy in the ephemeral configuration database.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
23.4R1-EVO	Starting in Junos OS Evolved Release 23.4R1, Junos OS Evolved supports configuring MSTP, RSTP, and VSTP in the ephemeral configuration database on supported devices.
23.4R1	Starting in Junos OS Release 23.4R1, to configure MSTP, RSTP, or VSTP in the ephemeral configuration database, you must first configure the `ephemeral-db-support` statement at the `[edit protocols layer2-control]` hierarchy level in the static configuration database.
23.2R2	Starting in Junos OS Release 23.2R2, to configure MSTP, RSTP, or VSTP in the ephemeral configuration database, you must first configure the `ephemeral-db-support` statement at the `[edit protocols layer2-control]` hierarchy level in the static configuration database.
23.2R1	Starting in Junos OS Release 23.2R1, Junos OS supports configuring MSTP, RSTP, and VSTP in the ephemeral configuration database on supported devices.

RELATED DOCUMENTATION

Understanding the Ephemeral Configuration Database | 310

Enable and Configure Instances of the Ephemeral Configuration Database

IN THIS SECTION

The ephemeral database is an alternate configuration database. It enables multiple client applications to concurrently load and commit configuration changes to a Junos device and with significantly greater throughput than when committing data to the candidate configuration database. Junos devices provide a default ephemeral database instance as well as the ability to enable and configure multiple user-defined instances of the ephemeral configuration database.

NETCONF and Junos XML protocol client applications and JET applications can update the ephemeral configuration database. The following sections detail how to enable instances of the ephemeral configuration database, configure the instances using NETCONF and Junos XML protocol operations, and display ephemeral configuration data in the CLI. The sections also discuss how to deactivate and then reactivate an ephemeral instance as well as delete an ephemeral instance. For information about using JET applications to configure the ephemeral configuration database, see the Juniper Extension Toolkit Documentation.

Enable Ephemeral Database Instances

The default ephemeral database instance is automatically enabled on Junos devices that support configuring the ephemeral database. However, you must configure any user-defined instances of the ephemeral configuration database before you can use the instance. See Feature Explorer to verify the hardware platforms and software releases that support the ephemeral database.

To enable a user-defined instance of the ephemeral configuration database:

1. Configure the name of the instance.

   The name must contain only alphanumeric characters, hyphens, and underscores, and it must not exceed 32 characters. You cannot use `default` as the name.

```
[edit system configuration-database ephemeral]
user@host# set instance instance-name
```

> **NOTE**: The priority of ephemeral database instances is determined by the order in which the configuration lists the instances. By default, newly configured instances are placed at the end of the list and have lower priority when resolving conflicting configuration statements. When you configure a new instance, you can specify its placement by using the `insert` command instead of the `set` command.

2. Commit the configuration.

```
[edit system configuration-database ephemeral]
user@host# commit
```

> **NOTE**: When you commit statements at the `[edit system configuration-database ephemeral]` hierarchy level, all Junos processes must check and evaluate their complete configuration. As a result, there might be a spike in CPU utilization, potentially impacting other critical software processes.

Configure Ephemeral Database Options

You can configure several options for the ephemeral configuration database. You configure the options in the static configuration database.

1. (Optional) To disable the default instance of the ephemeral configuration database, configure the `ignore-ephemeral-default` statement.

```
[edit system configuration-database ephemeral]
user@host# set ignore-ephemeral-default
```

2. (Optional) Configure the commit synchronize model as asynchronous or synchronous.

The synchronous commit model is slower, but it is more reliable when synchronizing ephemeral configuration data to a backup Routing Engine or Virtual Chassis member.

```
[edit system configuration-database ephemeral]
user@host# set commit-synchronize-model (asynchronous | synchronous)
```

NOTE: Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the default commit synchronize model is `synchronous` and devices that enable GRES must use the synchronous model. In earlier releases, the default is `asynchronous`.

3. (Optional) When the device has graceful Routing Engine switchover (GRES) enabled and the ephemeral database uses the asynchronous commit synchronize model, configure the `allow-commit-synchronize-with-gres` statement to enable the device to synchronize an ephemeral instance to the other Routing Engine when you request a commit synchronize operation on that instance.

```
[edit system configuration-database ephemeral]
user@host# set allow-commit-synchronize-with-gres
```

NOTE: Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, we've deprecated the `allow-commit-synchronize-with-gres` statement and only the synchronous commit synchronize model supports synchronizing ephemeral data on devices that enable GRES.

4. Commit the configuration.

```
[edit system configuration-database ephemeral]
user@host# commit
```

NOTE: When you commit statements at the `[edit system configuration-database ephemeral]` hierarchy level, all Junos processes must check and evaluate their complete configuration. As a result, there might be a spike in CPU utilization, potentially impacting other critical software processes.

Enable MSTP, RSTP, and VSTP Configuration

On supported devices and releases, you can configure the following protocols in the ephemeral configuration database:

- Multiple Spanning Tree Protocol (MSTP)
- Rapid Spanning Tree Protocol (RSTP)
- VLAN Spanning Tree Protocol (VSTP)

Junos OS Evolved supports configuring these protocols in the ephemeral database in supported releases by default. However, on devices running Junos OS, you must configure the device to enable support for these protocols in the ephemeral database.

To enable users to configure MSTP, RSTP, and VSTP in the ephemeral database on devices running Junos OS:

1. In the static configuration database, configure the `ephemeral-db-support` statement at the `[edit protocols layer2-control]` hierarchy level.

```
[edit protocols layer2-control]
user@host# set ephemeral-db-support
```

2. Commit the configuration.

```
[edit protocols layer2-control]
user@host# commit
```

Open Ephemeral Database Instances

A client application must open an ephemeral database instance before viewing or modifying it. Within a NETCONF or Junos XML protocol session, a client application opens the ephemeral database instance by using the Junos XML protocol `<open-configuration>` operation with the appropriate child tags. Opening the ephemeral instance automatically acquires an exclusive lock on it.

- To open the default instance of the ephemeral database, a client application emits the `<open-configuration>` element and includes the `<ephemeral/>` child tag.

```
<rpc>
    <open-configuration>
        <ephemeral/>
    </open-configuration>
</rpc>
```

- To open a user-defined instance of the ephemeral database, a client application emits the <open-configuration> element and includes the <ephemeral-instance> element and the instance name.

```
<rpc>
    <open-configuration>
        <ephemeral-instance>instance-name</ephemeral-instance>
    </open-configuration>
</rpc>
```

Configure Ephemeral Database Instances

Client applications update the ephemeral configuration database using NETCONF and Junos XML protocol operations. Only a subset of the operations' attributes and options are available for use when updating the ephemeral configuration database. For example, options and attributes that reference groups, interface ranges, or commit scripts, or that roll back the configuration cannot be used with the ephemeral database.

Client applications load and commit configuration data to an open instance of the ephemeral configuration database. A client can load configuration data in any of the supported formats including Junos XML elements, formatted ASCII text, set commands, or JSON. By default, if a client disconnects from a session or closes the ephemeral database instance before committing new changes, the device discards any uncommitted data, but configuration data that has already been committed to the ephemeral database instance by that client is unaffected.

To update, commit, and close an open instance of the ephemeral configuration database, client applications perform the following tasks:

1. Load configuration data into the ephemeral database instance by performing one or more load operations.

   Client applications emit the <load-configuration> operation in a Junos XML protocol session or the <load-configuration> or <edit-config> operation in a NETCONF session and include the appropriate attributes and tags for the data.

```
<rpc>
    <load-configuration action="(merge | override | replace | set | update)" format="(text |
json | xml)">
        <!--configuration-data-->
    </load-configuration>
</rpc>
```

NOTE: The ephemeral configuration database supports the `update` attribute starting in Junos OS Release 21.1R1.

NOTE: The only acceptable format for `action="set"` is `"text"`. For more information about the `<load-configuration>` operation, see "<load-configuration>" on page 130.

```
<rpc>
    <edit-config>
        <target>
            <candidate/>
        </target>
        <!--configuration-data-->
    </edit-config>
</rpc>
```

NOTE: The target value `<candidate/>` can refer to either the open configuration database, or if there is no open database, to the candidate configuration. If a client application issues the Junos XML protocol `<open-configuration>` operation to open an ephemeral instance before executing the `<edit-config>` operation, the device performs the `<edit-config>` operation on the open instance of the ephemeral configuration database. Otherwise, the device performs the operation on the candidate configuration.

2. (Optional) Review the updated configuration in the open ephemeral instance by emitting the `<get-configuration/>` operation in a Junos XML protocol session or the `<get-configuration/>` or `<get-config>` operation in a NETCONF session.

```
<rpc>
    <get-configuration format="(json | set | text | xml)"/>
</rpc>
```

```
<rpc>
    <get-config>
        <source>
            <candidate/>
        </source>
```

```
    </get-config>
</rpc>
```

3. Commit the configuration changes by emitting the <commit-configuration/> operation in a Junos XML protocol session or the <commit-configuration/> or <commit/> operation in a NETCONF session.

   Include the <synchronize/> tag in the <commit-configuration> element to synchronize the data to a backup Routing Engine or to other members of a Virtual Chassis.

```
<rpc>
    <commit-configuration/>
</rpc>
```

```
<rpc>
    <commit-configuration>
        <synchronize/>
    </commit-configuration>
</rpc>
```

```
<rpc>
    <commit/>
</rpc>
```

NOTE: Starting in Junos OS Release 22.1R1, you can automatically synchronize an ephemeral instance's configuration to the other Routing Engine every time you commit the instance. To automatically synchronize an instance, include the `synchronize` statement at the `[edit system commit]` hierarchy level within that ephemeral instance's configuration.

NOTE: After a client application commits changes to the ephemeral database instance, the device merges the ephemeral data into the active configuration according to the rules of prioritization.

4. Repeat steps 1 through 3 for any subsequent updates to the ephemeral database instance.

5. Close the ephemeral database instance, which releases the exclusive lock.

```
<rpc>
    <close-configuration/>
</rpc>
```

Display Ephemeral Configuration Data in the CLI

The active device configuration is a merged view of the static and ephemeral configuration databases. However, when you display the configuration using the `show configuration` command in operational mode, the output does not include ephemeral configuration data. To display the data in a specific ephemeral database instance or display a merged view of the static and ephemeral configuration databases, use variations of the `show ephemeral-configuration` CLI command.

Table 16 on page 335 summarizes the `show ephemeral-configuration` commands.

Table 16: `show ephemeral-configuration` Command

Action	`show ephemeral-configuration` Command
View the configuration data in the default ephemeral instance.	**show ephemeral-configuration instance default**
View the configuration data in a user-defined ephemeral instance.	**show ephemeral-configuration instance *instance-name***
View the complete post-inheritance configuration merged with the configuration data in all instances of the ephemeral database.	**show ephemeral-configuration merge**
Specify the scope of the configuration data to display in a specific ephemeral instance. Append the statement path of the requested hierarchy to the command.	**show ephemeral-configuration instance *instance-name hierarchy-to-view*** For example: **show ephemeral-configuration instance default protocols mpls**

Deactivate Ephemeral Database Instances

When you enable and configure an ephemeral instance, the Junos device stores the instance's configuration data in files, which is similar to the operation of the static configuration database. You can deactivate a specific ephemeral instance within the static configuration database. When you deactivate an instance and commit the configuration, the device preserves the instance's configuration data and files, but it does not merge the instance's configuration with the static configuration database. If you later reactivate the instance in the static configuration database, the device merges the instance's existing configuration data with the static configuration database.

NOTE: On devices running Junos OS Release 22.1R1 or later and devices running Junos OS Evolved, when you deactivate the entire `[edit system configuration-database ephemeral]` hierarchy level and commit the configuration, the device deletes the files and corresponding configuration data for all user-defined ephemeral instances. In earlier Junos OS releases, the device preserves the files and configuration data; however, the device does not merge the configuration data with the static configuration database. Deactivating the hierarchy does not affect the default ephemeral instance's files.

To deactivate the default ephemeral instance or a user-defined ephemeral instance in the static configuration database:

1. Deactivate the ephemeral database instance.
   - Deactivate the default ephemeral instance by configuring the `ignore-ephemeral-default` statement.

     ```
     [edit system configuration-database ephemeral]
     user@host# set ignore-ephemeral-default
     ```

   - Deactivate a user-defined ephemeral instance by issuing the `deactivate` command and specifying the instance name.

     ```
     [edit system configuration-database ephemeral]
     user@host# deactivate instance instance-name
     ```

2. Commit the configuration.

   ```
   [edit system configuration-database ephemeral]
   user@host# commit
   ```

To reactivate an ephemeral instance and thus merge its configuration with the static configuration database again:

1. Activate the ephemeral database instance.

   - Activate the default ephemeral instance by deleting the `ignore-ephemeral-default` statement.

     ```
     [edit system configuration-database ephemeral]
     user@host# delete ignore-ephemeral-default
     ```

   - Activate a user-defined ephemeral instance by issuing the `activate` command and specifying the instance name.

     ```
     [edit system configuration-database ephemeral]
     user@host# activate instance instance-name
     ```

2. Commit the configuration.

   ```
   [edit system configuration-database ephemeral]
   user@host# commit
   ```

Delete Ephemeral Database Instances

When you enable and configure an ephemeral instance, the Junos device stores the instance's configuration data in files, which is similar to the operation of the static configuration database. On devices running Junos OS Release 22.1R1 or later and devices running Junos OS Evolved, when you delete an ephemeral instance from the static configuration database and commit the configuration, the device also deletes the ephemeral instance's files and corresponding configuration data. Thus, if you later configure an ephemeral instance with the same name, there is no existing configuration data associated with this instance name.

However, in earlier Junos OS releases, when you delete an ephemeral instance, the device preserves the ephemeral instance's files. Thus, if you later configure an ephemeral instance with the same name, the device restores the configuration data associated with the instance name from the corresponding files. If you delete an ephemeral instance in an earlier release, we recommend that you delete the ephemeral instance's configuration data before you delete the instance from the static configuration database.

To delete the default ephemeral instance or a user-defined ephemeral instance from the static configuration database:

1. Delete the ephemeral database instance.

- Delete the default ephemeral instance by configuring the `delete-ephemeral-default` and `ignore-ephemeral-default` statements.

```
[edit system configuration-database ephemeral]
user@host# set delete-ephemeral-default
user@host# set ignore-ephemeral-default
```

> **NOTE**: Devices running Junos OS Release 22.1R1 or later and devices running Junos OS Evolved support the `delete-ephemeral-default` statement.

- Delete a user-defined ephemeral instance by issuing the `delete` command and specifying the instance name.

```
[edit system configuration-database ephemeral]
user@host# delete instance instance-name
```

2. Commit the configuration.

```
[edit system configuration-database ephemeral]
user@host# commit
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the default commit synchronize model is `synchronous`. In earlier releases, the default is `asynchronous`.
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES must use the synchronous commit synchronize model.
22.1R1	Starting in Junos OS Release 22.1R1, when you deactivate the entire `[edit system configuration-database ephemeral]` hierarchy level, Junos OS deletes the files and corresponding configuration data for all user-defined ephemeral instances. In earlier releases, the files and configuration data are preserved; however, the configuration data is not merged with the static configuration database.

22.1R1	Starting in Junos OS Release 22.1R1, when you delete an ephemeral instance in the static configuration database, the instance's configuration files are also deleted. In earlier releases, the configuration files are preserved.
18.2R1	Starting in Junos OS Release 18.2R1, the `show ephemeral-configuration` operational mode command uses a different syntax and supports filtering for displaying specific hierarchy levels.
18.1R1	Starting in Junos OS Release 18.1R1, the ephemeral configuration database supports loading configuration data using the `<load-configuration> action` attribute values of `override` and `replace` in addition to the previously supported values of `merge` and `set`.

RELATED DOCUMENTATION

Example: Configuring the Ephemeral Configuration Database Using NETCONF

Understanding the Ephemeral Configuration Database | **310**

Commit and Synchronize Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol | **339**

ephemeral

show ephemeral-configuration

Commit and Synchronize Ephemeral Configuration Data Using the NETCONF or Junos XML Protocol

IN THIS SECTION

Commit an Ephemeral Instance Overview

The ephemeral database is an alternate configuration database. It enables NETCONF and Junos XML protocol client applications to simultaneously load and commit configuration changes on Junos devices and with significantly greater throughput than when committing data to the candidate configuration database. Client applications can commit the configuration data in an open instance of the ephemeral configuration database so that it becomes part of the active configuration on the device. When you commit ephemeral configuration data on a device, the device's active configuration is a merged view of the static and ephemeral configuration databases.

CAUTION: The ephemeral commit model validates the syntax but not the semantics, or constraints, of the configuration data committed to the ephemeral database. You must validate all configuration data before loading it into the ephemeral database and committing it on the device. Committing invalid configuration data can cause Junos processes to restart or stop responding and result in disruption to the system or network.

After a client application commits an ephemeral instance, the device merges the configuration data into the ephemeral database. The system processes parse the configuration and then merge the ephemeral data with the data in the active configuration. If there are conflicting statements in the static and ephemeral configuration databases, the device merges the data according to specific rules of prioritization. The database priority, from highest to lowest, is as follows:

1. Statements in a user-defined instance of the ephemeral configuration database.

   If the device uses multiple user-defined ephemeral instances, it determines the priority by the order in which the instances are configured at the `[edit system configuration-database ephemeral]` hierarchy level, running from highest to lowest priority.

2. Statements in the default ephemeral database instance.

3. Statements in the static configuration database.

NOTE: Applications can simultaneously load and commit data to different ephemeral database instances in addition to the static configuration database. However, the device processes the commits sequentially. As a result, the commit to a specific database might be delayed, depending on the processing order.

NOTE: If you commit ephemeral configuration data that is invalid or results in undesirable network disruption, you must remove the problematic data from the

database. You can delete the data, or if necessary, you can reboot the device, which deletes the configuration data in all instances of the ephemeral configuration database.

The active device configuration is a merged view of the static and ephemeral configuration databases. However, when you display the configuration using the `show configuration` command in operational mode, the output does not include ephemeral configuration data. To display the data in a specific ephemeral database instance or display a merged view of the static and ephemeral configuration databases, use variations of the `show ephemeral-configuration` CLI command.

How to Commit an Ephemeral Instance

Client applications can commit the configuration data in an open instance of the ephemeral configuration database so that it becomes part of the active configuration on the device. To commit the configuration, use the `<commit-configuration/>` operation in a Junos XML protocol session or the `<commit-configuration/>` or `<commit/>` operation in a NETCONF session.

In a Junos XML protocol session, a client application commits the configuration data in an open instance of the ephemeral configuration database by performing a `<commit-configuration/>` operation (just as for the candidate configuration).

```
<rpc>
    <commit-configuration/>
</rpc>
```

The Junos XML protocol server reports the results of the commit operation in `<rpc-reply>`, `<commit-results>`, and `<routing-engine>` tag elements. If the commit operation succeeds, the `<routing-engine>` element encloses the `<commit-success/>` tag and the `<name>` element, which specifies the target Routing Engine.

```
<rpc-reply xmlns:junos="URL">
   <commit-results>
       <routing-engine>
           <name>routing-engine-name</name>
           <commit-success/>
       </routing-engine>
   </commit-results>
</rpc-reply>
```

In a NETCONF session, a client application commits the configuration data in an open instance of the ephemeral configuration database by performing a <commit/> or <commit-configuration/> operation (just as for the candidate configuration).

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

```
<rpc>
    <commit-configuration/>
</rpc>
]]> ]]>
```

The NETCONF server confirms that the commit operation was successful by returning the <ok/> tag in an <rpc-reply> tag element.

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

If the commit operation fails, the NETCONF server returns the <rpc-reply> element and <rpc-error> child element, which explains the reason for the failure.

The only variant of the commit operation supported for the ephemeral database is synchronizing the configuration, as described in "Overview of Synchronizing an Ephemeral Instance" on page 342.

Overview of Synchronizing an Ephemeral Instance

Dual Routing Engine devices and Virtual Chassis systems do not automatically synchronize ephemeral configuration data when you commit an ephemeral instance. You can synchronize the data in an ephemeral instance on a per-commit or per-session basis. You can also configure an ephemeral instance to synchronize its data every time you commit the instance. The environment determines where the data is synchronized, for example:

- A dual Routing Engine device synchronizes the ephemeral instance to the backup Routing Engine.
- An MX Series Virtual Chassis synchronizes the ephemeral instance only to the backup device's primary Routing Engine.
- An EX Series Virtual Chassis synchronizes the ephemeral instance to all members switches.

NOTE: Virtual Chassis environments do not support synchronizing the ephemeral configuration database to the backup Routing Engine on the respective Virtual Chassis member.

See the following sections for instructions on synchronizing ephemeral instances:

- "How to Configure GRES-Enabled Devices to Synchronize Ephemeral Configuration Data" on page 345
- "How to Synchronize an Ephemeral Instance on a Per-Commit Basis" on page 346
- "How to Synchronize an Ephemeral Instance on a Per-Session Basis" on page 347
- "How to Automatically Synchronize an Ephemeral Instance upon Commit" on page 348
- "How to Configure Failover Configuration Synchronization for the Ephemeral Database" on page 349

The ephemeral database supports two commit synchronize models: asynchronous and synchronous. Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the ephemeral database uses the synchronous model by default. In earlier releases, the asynchronous model is the default. You can configure the `commit-synchronize-model` statement to explicitly configure the model.

```
[edit system configuration-database ephemeral]
user@host# set commit-synchronize-model (asynchronous | synchronous)
```

In the asynchronous commit model, the NETCONF or Junos XML protocol server first commits the configuration on the local Routing Engine and then notifies the other Routing Engine or Virtual Chassis device. The requesting Routing Engine does not wait for the other Routing Engine or Virtual Chassis member to first synchronize and commit the configuration.

Synchronous commit operations are slower but more reliable than asynchronous commit operations. We recommend that you use the synchronous commit model on devices that have graceful Routing Engine switchover (GRES) or nonstop active routing (NSR) enabled. In the synchronous model, the primary Routing Engine or MX Virtual Chassis primary device generally only completes its commit operation if the commit on the backup Routing Engine or Virtual Chassis backup device is successful.

When you synchronize an ephemeral instance, the Junos XML protocol server reports the results of the commit operation for the local Routing Engine in `<rpc-reply>`, `<commit-results>`, and `<routing-engine>` tag elements. If the commit operation succeeds, the `<routing-engine>` element encloses the `<commit-success/>` tag and the `<name>` element, which specifies the target Routing Engine.

The server reply includes additional tags that depend on the commit synchronize model used by the database.

- If the ephemeral database uses the synchronous model, the server reply includes a second <routing-engine> element for the commit operation on the other Routing Engine.
- If the ephemeral database uses the asynchronous model, the server includes the <commit-synchronize-server-success> element. This tag indicates that the synchronize operation is scheduled on the other Routing Engine or Virtual Chassis members and provides the estimated time in seconds required for the operation to complete.

For example:

```
<rpc-reply xmlns:junos="URL">
  <commit-results>
    <routing-engine>
      <name>re0</name>
      <commit-success/>
    </routing-engine>
  </commit-results>
  <commit-synchronize-server-success>
    <current-job-id>0</current-job-id>
    <number-of-jobs>1</number-of-jobs>
    <estimated-time>60</estimated-time>
  </commit-synchronize-server-success>
</rpc-reply>
```

For synchronous commit operations, the RPC reply indicates the success or failure of the commit operation on the other Routing Engine or Virtual Chassis members. For asynchronous commit operations, the device records the success or failure of the scheduled commit operations in the system log file. You must configure the device to log events of the given facility and severity level corresponding to these operations. See the System Log Explorer for the various ephemeral database events and the facility and severity levels required to log them.

Similarly, in NETCONF sessions, the server confirms that the commit operation was successful by returning the <ok/> tag in an <rpc-reply> tag element. Depending on the device configuration, the response might also include the <commit-results> element for synchronous commit synchronize operations or the <commit-synchronize-server-success> element for asynchronous commit synchronize operations. For example:

```
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <ok/>
  <commit-synchronize-server-success>
    <current-job-id>0</current-job-id>
    <number-of-jobs>1</number-of-jobs>
```

```
    <estimated-time>60</estimated-time>
  </commit-synchronize-server-success>
</rpc-reply>
]]>]]>
```

NOTE: The device does not synchronize the ephemeral configuration database to the other Routing Engine or Virtual Chassis members when you issue the `commit synchronize` command on the static configuration database.

How to Configure GRES-Enabled Devices to Synchronize Ephemeral Configuration Data

The ephemeral database supports two commit synchronize models: asynchronous and synchronous. To ensure a GRES-enabled device synchronizes ephemeral configuration data when you request a commit synchronize operation on an ephemeral instance, you must use one of the following methods:

- Use the synchronous model
- Use the asynchronous model and configure the `allow-commit-synchronize-with-gres` statement

We recommend that you use the synchronous model on devices that enable GRES. Additionally, starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the ephemeral database uses the synchronous model by default and devices that enable GRES must use the synchronous model. In earlier releases, the asynchronous model supports synchronizing ephemeral data on GRES-enabled devices provided that you configure the `allow-commit-synchronize-with-gres` statement. However, we do not recommend using the asynchronous model on devices that enable GRES.

To configure GRES-enabled devices to synchronize ephemeral configuration data:

1. Configure the commit model that the ephemeral database uses to perform commit synchronize operations.
   - (Recommended) To use the synchronous commit model, configure the `synchronous` option.

     ```
     [edit system configuration-database ephemeral]
     user@host# set commit-synchronize-model synchronous
     ```

- Alternatively, to use the asynchronous commit model in Junos OS Release 25.2 and earlier or Junos OS Evolved Release 25.2 and earlier, configure the `asynchronous` option and the `allow-commit-synchronize-with-gres` statement.

```
[edit system configuration-database ephemeral]
user@host# set commit-synchronize-model asynchronous
user@host# set allow-commit-synchronize-with-gres
```

2. Commit the configuration.

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

How to Synchronize an Ephemeral Instance on a Per-Commit Basis

You can synchronize an ephemeral instance across Routing Engines or Virtual Chassis members for a given commit operation on that instance.

To synchronize an ephemeral instance on a per-commit basis:

1. Open the ephemeral instance.

```
<rpc>
   <open-configuration>
      <ephemeral-instance>instance-name</ephemeral-instance>
   </open-configuration>
</rpc>
```

2. Configure the ephemeral instance.

```
<rpc>
    <load-configuration>
        <!--configuration-data-->
    </load-configuration>
</rpc>
```

3. Commit and synchronize the instance by enclosing the empty <synchronize/> tag in the <commit-configuration> and <rpc> tag elements.

```
<rpc>
    <commit-configuration>
        <synchronize/>
    </commit-configuration>
</rpc>
```

4. Repeat steps 2 and 3, as appropriate.
5. Close the ephemeral instance.

```
<rpc>
   <close-configuration/>
</rpc>
```

How to Synchronize an Ephemeral Instance on a Per-Session Basis

You can synchronize an ephemeral instance across Routing Engines or Virtual Chassis members for all commit operations performed for the duration that the ephemeral instance is open, which we are loosely referring to as a session. This session should not be confused with the NETCONF or Junos XML protocol session. Synchronizing the instance on a per-session basis enables you to execute multiple load and commit operations and ensure that each commit operation automatically synchronizes the instance until you close it.

To synchronize an ephemeral instance for all commit operations performed for the duration that the instance is open:

1. Open the ephemeral instance, and include the <commit-synchronize/> tag.

```
<rpc>
    <open-configuration>
        <ephemeral-instance>instance-name</ephemeral-instance>
        <commit-synchronize/>
    </open-configuration>
</rpc>
```

2. Configure the ephemeral instance.

```
<rpc>
    <load-configuration>
```

```
        <!--configuration-data-->
    </load-configuration>
</rpc>
```

3. Commit the instance, which also synchronizes it to the other Routing Engine or Virtual Chassis members.

```
<rpc>
    <commit-configuration/>
</rpc>
```

4. Repeat steps 2 and 3, as appropriate.
5. Close the ephemeral instance.

```
<rpc>
    <close-configuration/>
</rpc>
```

How to Automatically Synchronize an Ephemeral Instance upon Commit

On devices running Junos OS Release 22.1R1 or later and devices running Junos OS Evolved, you can configure an ephemeral instance so that it synchronizes its configuration across Routing Engines or Virtual Chassis members every time you commit the instance.

To configure the ephemeral instance to synchronize every time you commit the instance:

1. Open the ephemeral instance.

```
<rpc>
   <open-configuration>
      <ephemeral-instance>instance-name</ephemeral-instance>
   </open-configuration>
</rpc>
```

2. Configure the ephemeral instance to include the `synchronize` statement at the `[edit system commit]` hierarchy level.

```
<rpc>
    <load-configuration>
        <configuration>
            <system>
```

```
                <commit>
                    <synchronize/>
                </commit>
            </system>
        </configuration>
    </load-configuration>
</rpc>
```

3. Commit the instance, which also synchronizes its configuration to the other Routing Engine.

```
<rpc>
    <commit-configuration/>
</rpc>
```

4. Close the ephemeral instance.

```
<rpc>
    <close-configuration/>
</rpc>
```

After you add the `synchronize` statement at the `[edit system commit]` hierarchy level in the ephemeral instance's configuration, the device automatically synchronizes the instance to the other Routing Engine or Virtual Chassis members whenever you commit that instance, provided that the device meets the necessary requirements for synchronizing the database.

How to Configure Failover Configuration Synchronization for the Ephemeral Database

Dual Routing Engine devices and MX Series Virtual Chassis support failover configuration synchronization for the ephemeral database. Failover configuration synchronization helps ensure that the configuration database is synchronized between Routing Engines in the event of a Routing Engine switchover. To enable failover synchronization, you configure the `commit synchronize` statement at the `[edit system]` hierarchy level in the static configuration database.

When you configure the `commit synchronize` statement in the static configuration database, it has the following effects:

- The device synchronizes its static configuration database to the backup Routing Engine or MX Virtual Chassis backup device during a commit operation.

> **NOTE**: If you configure the `commit synchronize` statement in the static configuration database, the device does not automatically synchronize an ephemeral instance to the

backup device when you commit the static configuration database or when you commit the instance.

- Starting in Junos OS Release 20.2R1, a backup Routing Engine or an MX Virtual Chassis backup device synchronizes both its static and ephemeral configuration databases when it synchronizes with the primary device. In earlier releases, a backup Routing Engine only synchronizes the static configuration database.

  NOTE: For failover synchronization, the backup Routing Engine and the MX Virtual Chassis backup device only synchronize the ephemeral configuration database with the primary device if both the backup device and the primary device are running the same software version.

When you configure the `commit synchronize` statement on the primary and backup Routing Engines, the backup Routing Engine synchronizes its configuration with the primary Routing Engine in the following scenarios:

- You remove and reinsert the backup Routing Engine
- You reboot the backup Routing Engine
- The device performs a graceful Routing Engine switchover
- There is a manual change in roles
- You insert a new backup Routing Engine that has the `commit synchronize` statement configured

On a dual Routing Engine system, the backup Routing Engine synchronizes its configuration databases with the primary Routing Engine. In an MX Series Virtual Chassis, the primary Routing Engine on the backup device synchronizes its configuration databases with the primary Routing Engine on the primary device.

To enable failover configuration synchronization for both the static and ephemeral databases on supported devices running Junos OS Release 20.2R1 or later or devices running Junos OS Evolved:

1. Configure the `synchronize` statement in the static configuration database.

```
[edit]
user@host# set system commit synchronize
```

2. Commit the configuration.

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

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, the default commit synchronize model is `synchronous`. In earlier releases, the default is `asynchronous`.
25.4R1 & 25.4R1-EVO	Starting in Junos OS Release 25.4R1 and Junos OS Evolved Release 25.4R1, devices that enable GRES or NSR must use the synchronous commit synchronize model.
20.2R1	Starting in Junos OS Release 20.2R1, when you configure the `synchronize` statement at the `[edit system commit]` hierarchy level in the static configuration database, the backup Routing Engine synchronizes both the static and ephemeral configuration databases when it synchronizes with the primary Routing Engine. In earlier releases, the backup Routing Engine only synchronizes the static configuration database.

RELATED DOCUMENTATION

Enable and Configure Instances of the Ephemeral Configuration Database | 328

Understanding the Ephemeral Configuration Database | 310

Managing Ephemeral Configuration Database Space

SUMMARY

Configure options for ephemeral database instances to more effectively manage the amount of space that the database uses.

IN THIS SECTION

Junos devices maintain versions of ephemeral configuration database objects with every commit. Thus, any change to the ephemeral database, whether it is an addition, modification, or deletion, increases the size of the database. As a result, the database only increases in size over time. Depending on the size of the ephemeral configuration and the changes to the database, the database can consume a lot of disk space, become fragmented, and could potentially run into the maximum database size. You can manage the space that an ephemeral database instance uses by configuring different options.

In supported releases, Junos devices, by default, perform cyclic versioning when you commit an ephemeral instance. Cyclic versioning reclaims the space occupied by objects deleted in a previous database version. To manage the space consumed by the ephemeral database, you can configure the device to:

- Adjust cyclic versioning as appropriate for your operations.
- Resize an ephemeral database when it meets specific criteria.

Benefits of Cyclic Versioning and Resizing

- More efficiently manage ephemeral configuration database space as required for a given environment.
- Reduce database fragmentation for improved performance.
- Prevent an ephemeral configuration database from running into the maximum database size.

Understanding Cyclic Versioning

Junos devices maintain versioning for ephemeral database objects, and as a result, the database also retains and stores deleted objects. A deletion is characterized by:

- Explicitly deleting the configuration.
- Changing the value of a configuration attribute.
- Reordering elements during a load update operation.

Cyclic versioning reclaims the space occupied by objects that were deleted in a previous version of the database. The cyclic version value determines the ephemeral database version in which the system reclaims deleted objects during a commit operation. The default cyclic version value for each ephemeral database instance is 10. Thus, on devices that support cyclic versioning, the system, by default, reclaims the space occupied by deleted configuration objects with each commit. You can modify the setting on a per-instance basis. To disable cyclic versioning, set the cyclic version value to 0.

For example, if you use the default cyclic version value of 10, then:

- After the 11th commit (version 11), the device reclaims the space occupied by objects that were deleted in version 1.
- After the 12th commit (version 12), the device reclaims the space occupied by objects that were deleted in version 2.
- After the 13th commit (version 13), the device reclaims the space occupied by objects that were deleted in version 3.

This process continues with each subsequent commit operation. As illustrated in the previous example, the version from which the system reclaims deleted objects during the current commit operation is:

```
version to reclaim = current version - cyclic version
```

NOTE: When the system resizes the database, the system keeps only the active configuration objects and resets the version for each object to the latest version. As a result, the system does not reclaim deleted objects again until after you execute commit operations equal to the cyclic version value.

In earlier releases and on devices that do not use cyclic versioning, the ephemeral database default behavior is to purge the database when it reaches the maximum allowable version. A purge operation reclaims the space used by deleted objects but requires all processes to read the full configuration. A database purge operation involves:

- Creating a new database.
- Copying only the active configuration objects from the current database into the new database.
- Setting the version for all active configuration objects in the new database to version 1.

Understanding Ephemeral Database Resizing

Resizing an ephemeral database might be necessary if cyclic versioning is enabled and you make frequent changes to the database that involve deleting or reordering elements. On devices that support cyclic versioning, the system automatically reclaims the space occupied by deleted objects during a

commit operation. However, the system might or might not reallocate the freed space for new configuration objects when you update the database. If the system does not reallocate the space, then the database can become fragmented over time. Resizing an ephemeral database reclaims the space occupied by all deleted objects and defragments the database, which can improve performance.

A database resize operation involves:

- Creating a new database.
- Copying only the active configuration objects from the current database into the new database.
- Setting the version for all active configuration objects in the new database to the latest version.

As with the static configuration database, you can configure Junos devices to resize the ephemeral configuration database. After you configure database resizing, Junos devices resize the ephemeral database during a commit operation if the database's space exceeds the specified thresholds. You can modify the thresholds for each ephemeral instance.

The system resizes the database when the database size meets the criteria for both of the following configuration statements:

- `database-size-diff`—Minimum difference between the database size and the actual usage. Default is 100 MB.
- `database-size-on-disk`—Minimum configuration database size on disk. Default is 450 MB.

For example, suppose you configure the device to use the default values. Then the system resizes the database when the database size on disk exceeds 450 MB *and* the database size is 100 MB greater than the actual database usage.

For information about configuring database resizing, see "Resize an Ephemeral Database Instance" on page 356.

Use the `show system configuration database usage` command to display the database's disk space usage. The command displays the current database size on disk, the actual database usage, and the maximum size of the database.

```
user@host> show system configuration database usage ephemeral-instance default
Maximum size of the database: 692.49 MB
Current database size on disk: 1.50 MB
Actual database usage: 1.49 MB
Available database space: 691.01 MB
```

Configure Cyclic Versioning

Junos devices, by default, use a cyclic version value of 10. When configuring the cyclic version value, the best practice is to use a smaller value if you perform frequent commit operations for scaled configurations that reorder elements or delete many objects or attributes. A smaller value causes the device to store deleted objects for fewer versions of the database and thus use less disk space overall. In such cases, we recommend a value of 2 or 3. Otherwise, you can use a larger cyclic version value, such as the default value of 10.

NOTE: If a Junos process misses reading more commits than the configured cyclic version value, it must read the full configuration because the delta between the versions is no longer available. This effect might happen more frequently if you configure smaller cyclic version values.

To specify the cyclic version value that the device uses to reclaim the space occupied by deleted objects during a commit operation:

1. Configure the cyclic version value for the default ephemeral instance.

```
[edit system configuration-database ephemeral]
user@host# set cyclic-version-for-ephemeral-default version
```

For example:

```
[edit system configuration-database ephemeral]
user@host# set cyclic-version-for-ephemeral-default 8
```

2. Configure the cyclic version value for a user-defined ephemeral instance.

```
[edit system configuration-database ephemeral]
user@host# set instance instance-name cyclic-version verison
```

For example:

```
[edit system configuration-database ephemeral]
user@host# set instance eph1 cyclic-version 3
```

3. Commit the configuration.

```
[edit system configuration-database ephemeral]
user@host# commit
```

Resize an Ephemeral Database Instance

Junos devices do not automatically resize an ephemeral database. You can configure the device to resize an ephemeral database during a commit operation when the database size meets certain thresholds. You can enable resizing and use either the default values or custom values that are appropriate for your environment. To configure resizing:

1. Enable resizing for the default ephemeral instance.

   - To use the default values, configure the top-level `resize-ephemeral-default` statement.

     ```
     [edit system configuration-database ephemeral]
     user@host# set resize-ephemeral-default
     ```

   - To use custom values, configure the database size difference and the database size on disk in MB.

     ```
     [edit system configuration-database ephemeral]
     user@host# set resize-ephemeral-default database-size-diff size
     user@host# set resize-ephemeral-default database-size-on-disk size
     ```

     For example:

     ```
     [edit system configuration-database ephemeral]
     user@host# set resize-ephemeral-default database-size-diff 50
     user@host# set resize-ephemeral-default database-size-on-disk 600
     ```

2. Enable resizing for a user-defined instance.

   - To use the default values, configure the top-level `resize` statement.

     ```
     [edit system configuration-database ephemeral]
     user@host# set instance instance-name resize
     ```

- To use custom values, configure the database size difference and the database size on disk in MB.

```
[edit system configuration-database ephemeral]
user@host# set instance instance-name resize database-size-diff size
user@host# set instance instance-name resize database-size-on-disk size
```

For example:

```
[edit system configuration-database ephemeral]
user@host# set instance eph1 resize database-size-diff 150
user@host# set instance eph1 resize database-size-on-disk 500
```

3. Commit the configuration.

```
[edit system configuration-database ephemeral]
user@host# commit
```

After you configure the device to resize the database, the device resizes the database after a commit operation on that database when it meets the specified criteria. After successfully resizing the database, the device emits the following message:

```
Database resize completed
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
23.2R1 and 23.2R1-EVO	Starting in Junos OS Release 23.2R1 and Junos OS Evolved Release 23.2R1, Junos devices automatically perform cyclic versioning for the ephemeral configuration database. In earlier releases, the device purges deleted objects from the database only when it reaches the maximum version number.

RELATED DOCUMENTATION

ephemeral

4

PART

Request Operational and Configuration Information Using the Junos XML Protocol

CHAPTER 14

Request Operational Information Using the Junos XML Protocol

IN THIS CHAPTER

Request Operational Information Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use Junos XML request tags to request operational information from Junos devices.

A Junos XML protocol client application can request information about the current status of a device running Junos OS or a device running Junos OS Evolved. To request operational information, a client application emits an <rpc> element enclosing the specific Junos XML API request tag element that returns the desired information.

Table 17 on page 359 provides examples of request tags, which request the same information as the equivalent CLI command.

Table 17: Examples of Request Tags and Equivalent CLI Command

Request Tag	CLI Command
<get-interface-information>	show interfaces

Table 17: Examples of Request Tags and Equivalent CLI Command *(Continued)*

Request Tag	CLI Command
<get-chassis-inventory>	show chassis hardware
<get-system-inventory>	show software information

You can determine the appropriate Junos XML request tag using multiple methods, including:

- Appending `| display xml rpc` to an operational command in the CLI.
- Using the Junos XML API Explorer - Operational Tags application to search for a command or request tag in a given release.

For example, the following command displays the request tag corresponding to the `show interfaces` command:

```
user@router> show interfaces | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/23.4R1.9/junos">
    <rpc>
        <get-interface-information>
        </get-interface-information>
    </rpc>
</rpc-reply>
```

To execute an RPC, the client application encloses a request tag in an <rpc> element. The syntax depends on whether the request requires any additional command 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 format of the information returned by the Junos XML protocol server. By setting the optional `format` attribute in the opening operational request tag, a client application

can request the response in XML (the default), formatted ASCII text, or JavaScript Object Notation (JSON).

> **NOTE**: When displaying operational or configuration data that contains characters outside the 7-bit ASCII character set, the Junos device escapes and encodes these character using the equivalent UTF-8 decimal character reference. For more information see *How Character Encoding Works on Juniper Networks Devices*.

The Junos XML protocol server returns its reply in an `<rpc-reply>` element. The enclosed data is determined by the requested format.

If the client application requests XML output, the Junos XML protocol server encloses its response in the specific response tag element that corresponds to the request tag element.

```
<rpc-reply xmlns:junos="URL">
    <operational-response xmlns="URL-for-DTD">
        <!-- Junos XML tag elements for the requested information -->
    </operational-response>
</rpc-reply>
```

For example, if the client application sends the `<get-interface-information>` RPC, the server returns the `<interface-information>` response tag.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/23.4R1.9/junos">
<interface-information xmlns="http://xml.juniper.net/junos/23.4R1.9/junos-interface"
junos:style="normal">
<physical-interface>
<name>ge-0/0/0</name>
<admin-status junos:format="Enabled">up</admin-status>
<oper-status>up</oper-status>
...
```

For XML format, the opening tag for each operational response includes the `xmlns` attribute. The attribute defines the XML namespace for the enclosed tag elements that do not have a namespace prefix (such as `junos:`). 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
```

where:

- *release-code* is the standard string that represents the Junos OS release that is running on the Junos XML protocol server device.
- *category* specifies the DTD.

If the client application requests formatted ASCII text, the Junos XML protocol server encloses its response in an <output> element.

```
<rpc-reply xmlns:junos="URL">
    <output>
        operational-response
    </output>
</rpc-reply>
```

If the client application requests JSON format, the Junos XML protocol server encloses the JSON data in the <rpc-reply> tag element.

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

RELATED DOCUMENTATION

Understanding the Request Procedure in a Junos XML Protocol Session | 57

Request Configuration Data Using the Junos XML Protocol | 375

Specify the Output Format for Operational Information Requests in a Junos XML Protocol Session

SUMMARY

A Junos XML protocol client application can include the `format` attribute in Junos XML request tags to specify the output format for operational information requests on Junos devices.

A Junos XML protocol client application can request operational information about a device running Junos OS or a device running Junos OS Evolved. The client application emits an `<rpc>` element that encloses a Junos XML request tag element. The client application can request that the server return the output in the following formats:

- Junos XML elements (default)
- Formatted ASCII text
- JavaScript Object Notation (JSON)

To request the output in a specific format, the client application includes the optional `format` attribute in the opening operational request tag. The basic syntax is as follows:

```
<rpc>
    <operational-request format="(ascii | json | text | xml)">
        <!-- tag elements for options -->
    </operational-request>
</rpc>
```

The following sections outline how to request a specific format and the response from the server.

XML Format

By default, the Junos XML protocol server returns operational information in XML format. If the client application includes the `format="xml"` attribute, or if the application omits the `format` attribute, the server

returns the response in XML. The following example requests information for the ge-0/3/0 interface and omits the `format` attribute.

```
<rpc>
     <get-interface-information>
          <brief/>
          <interface-name>ge-0/3/0</interface-name>
     </get-interface-information>
</rpc>
```

The Junos XML protocol server returns the response in XML, which is identical to the output displayed in the CLI when you issue an operational mode command and include the `| display xml` option.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/24.4R1/junos">
<interface-information
 xmlns="http://xml.juniper.net/junos/24.4R1/junos-interface" junos:style="brief">
        <physical-interface>
            <name>ge-0/3/0</name>
            <admin-status junos:format="Enabled">up</admin-status>
            <oper-status>up</oper-status>
            <link-level-type>Ethernet</link-level-type>
            <mtu>1514</mtu>
            <sonet-mode>LAN-PHY</sonet-mode>
            <mru>1522</mru>
            <source-filtering>disabled</source-filtering>
            <speed>1000mbps</speed>
            <eth-switch-error>none</eth-switch-error>
            <remote-bounce>none</remote-bounce>
            <bpdu-error>none</bpdu-error>
            <ld-pdu-error>none</ld-pdu-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>
            <pad-to-minimum-frame-size>Disabled</pad-to-minimum-frame-size>
            <if-device-flags>
                <ifdf-present/>
                <ifdf-running/>
            </if-device-flags>
            <ifd-specific-config-flags>
```

```
                <internal-flags>0x100000</internal-flags>
            </ifd-specific-config-flags>
            <if-config-flags>
                <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>
```

ASCII Text

To request that the Junos XML protocol server return operational information as formatted ASCII text, the client application includes the `format="text"` or `format="ascii"` attribute in the opening request tag.

```
<rpc>
     <get-interface-information format="(text | ascii)">
          <brief/>
          <interface-name>ge-0/3/0</interface-name>
     </get-interface-information>
</rpc>
```

The Junos XML protocol server formats the reply as ASCII text and encloses it in an `<output>` element. The `format="text"` and `format="ascii"` attributes produce identical output.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/24.4R1/junos">
<output>
Physical interface: ge-0/3/0, Enabled, Physical link is Up
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, LAN-PHY mode, Speed: 1000mbps, Loopback:
Disabled,
  Source filtering: Disabled, Flow control: Enabled, Auto-negotiation: Enabled, Remote fault:
Online
  Device flags   : Present Running
  Interface Specific flags: Internal: 0x100000
  Interface flags: 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 Up
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, LAN-PHY mode, Speed: 1000mbps, Loopback:
Disabled,
  Source filtering: Disabled, Flow control: Enabled, Auto-negotiation: Enabled, Remote fault:
Online
  Device flags   : Present Running
  Interface Specific flags: Internal: 0x100000
  Interface flags: SNMP-Traps Internal: 0x4000
  Link flags     : None
```

The formatted ASCII text returned by the Junos XML protocol 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 Junos XML protocol 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 Junos XML protocol server returns the reply as formatted ASCII text enclosed in an `<output>` tag element even if XML-tagged output is requested.

NOTE: The content and formatting of data within an `<output>` tag element are subject to change, so client applications must not depend on them.

JSON Format

To request that the Junos XML protocol server return operational information in JSON format, the client application includes the `format="json"` attribute in the opening request tag.

```
<rpc>
     <get-interface-information format="json">
          <brief/>
          <interface-name>cbp0</interface-name>
     </get-interface-information>
</rpc>
```

The Junos XML protocol server returns JSON data.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/24.4R1/junos">
{
 "interface-information" : [
 {
    "attributes" : {"xmlns" : "http://xml.juniper.net/junos/24.4R1/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" : [
        {
            "internal-flags" : [
            {
                "data" : "0x0"
            }
            ]
        }
        ],
        "if-config-flags" : [
        {
            "iff-snmp-traps" : [
            {
                "data" : [null]
            }
            ]
        }
        ]
    }
```

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

By default, Junos devices emit JSON-formatted state data in non-compact format, which emits all objects as JSON arrays. In Junos OS Release 24.2 and earlier and Junos OS Evolved Release 24.2 and earlier, Junos devices support emitting the device's operational state in compact JSON format, in which only objects that have multiple values are emitted as JSON arrays. To emit compact JSON format in supported releases, configure the `compact` statement at the `[edit system export-format state-data json]` hierarchy level.

```
user@host# set system export-format state-data json compact
user@host# commit
```

The following example executes the `show system uptime | display json` command and displays the output in non-compact and compact JSON format.

Non-compact JSON format:

```
{
    "system-uptime-information" : [
    {
        "attributes" : {"xmlns" : "http://xml.juniper.net/junos/18.1R1/junos"},
        "current-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-15 13:43:46 PDT",
                "attributes" : {"junos:seconds" : "1526417026"}
            }
            ]
        }
        ],
        "time-source" : [
        {
            "data" : " NTP CLOCK "
        }
        ],
        "system-booted-time" : [
        {
```

```
            "date-time" : [
            {
                "data" : "2018-05-15 10:57:02 PDT",
                "attributes" : {"junos:seconds" : "1526407022"}
            }
            ],
            "time-length" : [
            {
                "data" : "02:46:44",
                "attributes" : {"junos:seconds" : "10004"}
            }
            ]
        }
        ],
        "protocols-started-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-15 10:59:33 PDT",
                "attributes" : {"junos:seconds" : "1526407173"}
            }
            ],
            "time-length" : [
            {
                "data" : "02:44:13",
                "attributes" : {"junos:seconds" : "9853"}
            }
            ]
        }
        ],
        "last-configured-time" : [
        {
            "date-time" : [
            {
                "data" : "2018-05-02 17:57:44 PDT",
                "attributes" : {"junos:seconds" : "1525309064"}
            }
            ],
            "time-length" : [
            {
                "data" : "1w5d 19:46",
                "attributes" : {"junos:seconds" : "1107962"}
            }
```

```
            ],
            "user" : [
            {
                "data" : "admin"
            }
            ]
        }
        ],
        "uptime-information" : [
        {
            "date-time" : [
            {
                "data" : "1:43PM",
                "attributes" : {"junos:seconds" : "1526417026"}
            }
            ],
            "up-time" : [
            {
                "data" : "2:47",
                "attributes" : {"junos:seconds" : "10034"}
            }
            ],
            "active-user-count" : [
            {
                "data" : "1",
                "attributes" : {"junos:format" : "1 user"}
            }
            ],
            "load-average-1" : [
            {
                "data" : "0.49"
            }
            ],
            "load-average-5" : [
            {
                "data" : "0.19"
            }
            ],
            "load-average-15" : [
            {
                "data" : "0.10"
            }
            ]
```

```
        }
        ]
    }
    ]
}
```

Compact JSON format:

```
{
    "system-uptime-information" :
    {
        "current-time" :
        {
            "date-time" : "2018-05-15 13:49:56 PDT"
        },
        "time-source" : " NTP CLOCK ",
        "system-booted-time" :
        {
            "date-time" : "2018-05-15 10:57:02 PDT",
            "time-length" : "02:52:54"
        },
        "protocols-started-time" :
        {
            "date-time" : "2018-05-15 10:59:33 PDT",
            "time-length" : "02:50:23"
        },
        "last-configured-time" :
        {
            "date-time" : "2018-05-15 13:49:40 PDT",
            "time-length" : "00:00:16",
            "user" : "admin"
        },
        "uptime-information" :
        {
            "date-time" : "1:49PM",
            "up-time" : "2:53",
            "active-user-count" : "1",
            "load-average-1" : "0.00",
            "load-average-5" : "0.06",
            "load-average-15" : "0.06"
        }
```

```
    }
}
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
24.4R1 & 24.4R1-EVO	Starting in Junos OS Release 24.4R1 and Junos OS Evolved Release 24.4R1, we've deprecated the `compact` statement at the `[edit system export-format state-data json]` hierarchy level.

RELATED DOCUMENTATION

Understanding the Request Procedure in a Junos XML Protocol Session | 57

Request Operational Information Using the Junos XML Protocol | 359

CHAPTER 15

Request Configuration Information Using the Junos XML Protocol

IN THIS CHAPTER

Request Configuration Data Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can use the <get-configuration> operation to request configuration data from Junos devices.

IN THIS SECTION

How to Request Configuration Data Using the Junos XML Protocol

A Junos XML protocol client application can request configuration data from a device running Junos OS or a device running Junos OS Evolved. To request configuration data, a client application encloses the <get-configuration> operation in an <rpc> element. A client application can include optional attributes to specify the source and format of the configuration information to return. In addition, a client application can define the scope of the information to return by including the appropriate child tag elements. For example, the client can request the entire configuration or specific portions of the configuration.

The basic syntax is as follows:

```
<rpc>
    <!-- Request the complete configuration -->
        <get-configuration [optional attributes]/>

     <!-- Request part of the configuration -->
        <get-configuration [optional attributes]>
            <configuration>
                <!-- tag elements representing the data to return -->
            </configuration>
        </get-configuration>
</rpc>
```

NOTE: To view configuration data in a specific instance of the ephemeral configuration database, a client application must open the ephemeral instance using the <open-configuration> operation with the appropriate child tags before executing the <get-configuration> request.

NOTE: A Junos XML protocol client application can use the <get-configuration> operation to request the entire logical system configuration or specific logical system configuration hierarchies.

The following topics describe how a client application specifies the source, format, and scope of information returned by the Junos XML protocol server:

- "Specify the Database Source for Configuration Data to Return" on page 377
- "Specify the Output Format for Configuration Data to Return" on page 382
- "Specify the Scope of the Configuration Data to Return" on page 387

The Junos XML protocol server encloses its reply in an <rpc-reply> element. Within the <rpc-reply> element, the server encloses the configuration data within the tag that corresponds to the requested format for all formats except JavaScript Object Notation (JSON).

- <configuration>—Encloses Junos XML-tagged output
- <configuration-text>—Encloses formatted ASCII output
- <configuration-set>—Encloses configuration mode commands

For Junos XML and JSON data, the server includes attributes with the junos: prefix to indicate when the configuration was last changed or committed and the user who committed it. For more information about the attributes, see "Specify the Database Source for Configuration Data to Return" on page 377.

```
<rpc-reply xmlns:junos="URL">
    <!-- If the application requests Junos XML tag elements -->
    <configuration junos:(changed | commit)-seconds="seconds" \
            junos:(changed | commit)-localtime="YYYY-MM-DD hh:mm:ss TZ" \
            [junos:commit-user="username"]>
        <!-- Junos XML tag elements representing configuration elements -->
    </configuration>

    <!-- If the application requests formatted ASCII text -->
    <configuration-text>
```

```
        <!-- formatted ASCII configuration statements -->
    </configuration-text>

    <!-- If the application requests configuration mode commands -->
    <configuration-set>
        <!-- configuration mode commands -->
    </configuration-set>

    <!-- If the application requests JSON format -->
    <!-- JSON configuration data -->
</rpc-reply>
```

NOTE: When displaying operational or configuration data that contains characters outside the 7-bit ASCII character set, Junos OS escapes and encodes these character using the equivalent UTF-8 decimal character reference. For more information see *Understanding Character Encoding on Devices Running Junos OS*.

Applications can also request other configuration-related information, including an XML schema representation of the configuration hierarchy or information about previously committed configurations. For more information, see the following:

- "Retrieve a Previous (Rollback) Configuration Using the Junos XML Protocol" on page 435
- "Retrieve the Rescue Configuration Using the Junos XML Protocol" on page 443
- "Compare the Active or Candidate Configuration to a Prior Version Using the Junos XML Protocol" on page 446
- "Compare Two Previous (Rollback) Configurations Using the Junos XML Protocol" on page 450
- "Request an XML Schema for the Configuration Hierarchy Using the Junos XML Protocol" on page 454

Specify the Database Source for Configuration Data to Return

IN THIS SECTION

A Junos XML protocol client application can use the <get-configuration> operation to request configuration data from a Junos device. A client application can request information from the following configuration databases:

- Candidate configuration database
- Active configuration database
- Open instance of the ephemeral configuration database

A client application can request information from the candidate configuration or the active configuration by using the <get-configuration> operation and setting the database attribute to the appropriate value. By default, the <get-configuration> operation returns data from the candidate configuration database. The Junos XML protocol server returns Junos XML-tagged output by default, except when the compare attribute is included.

The database attribute can be combined with one or more of the following attributes in the <get-configuration> tag: changed, commit-scripts, compare, format, inherit and optionally groups and interface-ranges.

The following sections: outline how to specify the source of the requested configuration data; describe the Junos XML protocol server reply; and provide a sample request.

Request Candidate Configuration Data

To request information from the candidate configuration database, a client application includes the database="candidate" attribute or omits the database attribute.

```
<rpc>
    <get-configuration/>

<!-- OR -->

    <get-configuration>
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

Request Active Configuration Data

To request information from the active configuration database, a client application includes the `database="committed"` attribute.

```
<rpc>
    <get-configuration database="committed"/>

<!-- OR -->

    <get-configuration database="committed">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

Request Ephemeral Configuration Data

To request information from a specific instance of the ephemeral configuration database, a client application first opens the ephemeral instance using the `<open-configuration>` operation with the appropriate child tags.

```
<rpc>
  <!-- Default instance  -->
    <open-configuration>
        <ephemeral/>
    </open-configuration>

  <!-- Named instance -->
    <open-configuration>
        <ephemeral-instance>instance-name</ephemeral-instance>
    </open-configuration>
</rpc>
```

After opening the ephemeral instance, the client application requests information from that instance by using the `<get-configuration>` operation. After all operations on the ephemeral instance are complete, the client application closes the instance with the `<close-configuration/>` operation.

```
<rpc>
    <get-configuration/>
```

```
  <!-- OR -->

    <get-configuration>
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>

<rpc>
    <close-configuration/>
</rpc>
```

Junos XML Protocol Server Reply

The Junos XML protocol server encloses its reply in an `<rpc-reply>` element. Depending on the database source, the Junos XML protocol server includes information about when the configuration was last changed or committed. Junos XML-tagged output and JSON data include this information as attributes in the opening `configuration` tag or object.

Table 18: `configuration` Attributes in RPC Reply

Database	Attribute	Description
Candidate configuration or open ephemeral instance	`junos:changed-localtime="`*`YYYY-MM-DD hh:mm:ss TZ`*`"`	Represents the time of the last change as the date and time in the device's local time zone.
	`junos:changed-seconds="`*`seconds`*`"`	Represents the time of the last change as the number of seconds since midnight on 1 January 1970.
Active configuration	`junos:commit-localtime="`*`YYYY-MM-DD hh:mm:ss TZ`*`"`	Represents the commit time as the date and time in the device's local time zone.
	`junos:commit-seconds="`*`seconds`*`"`	Represents the commit time as the number of seconds since midnight on 1 January 1970.
	`junos:commit-user="`*`username`*`"`	User who requested the commit operation.

For example, when returning information from the candidate configuration or from an instance of the ephemeral configuration database, the output includes information about when the configuration was last changed.

```
<rpc-reply xmlns:junos="URL">
    <configuration junos:changed-seconds="seconds" \
        junos:changed-localtime="YYYY-MM-DD hh:mm:ss TZ">
        <!-- Junos XML tag elements representing configuration elements -->
    </configuration>
</rpc-reply>
```

When returning information from the active configuration, the output includes information about when the configuration was last committed.

```
<rpc-reply xmlns:junos="URL">
    <configuration junos:commit-seconds="seconds" \
        junos:commit-localtime="YYYY-MM-DD hh:mm:ss TZ" \
        junos:commit-user="username">
        <!-- Junos XML tag elements representing configuration elements -->
    </configuration>
</rpc-reply>
```

Example: Request the Active Configuration

The following example requests the entire committed configuration. In actual output, the *Junos-version* variable is replaced by a value such as 20.4R1 for the initial version of Junos OS Release 20.4.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration database="committed"/>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration \
      junos:commit-seconds="seconds" \
      junos:commit-localtime="timestamp" \
      junos:commit-user="username">
    <version>Junos-version </version>
    <system>
      <host-name>big-router</host-name>
      <!- - other children of <system> - ->
    </system>
    <!- - other children of <configuration> - ->
  </configuration>
</rpc-reply>
```

T1185

Specify the Output Format for Configuration Data to Return

IN THIS SECTION

A Junos XML protocol client application can use the `<get-configuration>` operation to request configuration data from a Junos device. A client application can include the optional `format` attribute to specify the format of the configuration data. The application can request configuration data in the following formats:

- Junos XML format (default)
- Formatted ASCII text
- Configuration mode commands

- JSON

> **NOTE**: Client applications use Junos XML tag elements to represent the configuration elements to display regardless of which output format they request. The `format` attribute controls the format of the Junos XML protocol server's output only.

The following sections: outline how to specify the format of the requested configuration data; discuss the attributes that you can combine with the `format` attribute; and provide a sample request.

Request Junos XML Format

To request that the Junos XML protocol server return configuration data in Junos XML-tagged output, a client application includes the `format="xml"` attribute in the `<get-configuration>` tag or omits the `format` attribute. The Junos XML protocol server returns Junos XML-tagged output by default, except when the `compare` attribute is included.

```
<rpc>
        <get-configuration/>

    <!-- OR -->

        <get-configuration>
            <!-- tag elements for the configuration elements to return -->
        </get-configuration>
</rpc>
```

When the application requests Junos XML tag elements, the Junos XML protocol server encloses its output in `<rpc-reply>` and `<configuration>` elements.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- Junos XML tag elements representing configuration elements -->
    </configuration>
</rpc-reply>
```

Request Formatted ASCII Text

To request that the Junos XML protocol server return configuration data as formatted ASCII text, a client application includes the `format="text"` attribute in the `<get-configuration>` tag.

```
<rpc>
        <get-configuration format="text"/>

    <!-- OR -->

        <get-configuration format="text">
            <!-- tag elements for the configuration elements to return -->
        </get-configuration>
</rpc>
```

When the application requests formatted ASCII output, the Junos XML protocol server encloses the data in `<rpc-reply>` and `<configuration-text>` elements. The server formats the data in the same way that the CLI `show configuration` command displays configuration data. The output uses the newline character, tabs, braces, and square brackets to indicate the hierarchical relationships between configuration statements.

```
<rpc-reply xmlns:junos="URL">
    <configuration-text>
        <!-- formatted ASCII configuration statements -->
    </configuration-text>
</rpc-reply>
```

Request Configuration Mode Commands

To request configuration data formatted as configuration mode commands, a client application includes the `format="set"` attribute in the `<get-configuration>` tag.

```
<rpc>
        <get-configuration format="set"/>

    <!-- OR -->

        <get-configuration format="set">
            <!-- tag elements for the configuration elements to return -->
```

```
        </get-configuration>
</rpc>
```

When the application requests configuration mode commands, the Junos XML protocol server encloses the data in <rpc-reply> and <configuration-set> elements. The server formats the data in the same way that the CLI show configuration | display set command displays configuration data.

```
<rpc-reply xmlns:junos="URL">
    <configuration-set>
        <!-- configuration mode commands -->
    </configuration-set>
</rpc-reply>
```

Request JSON Format

To return configuration data in JSON format, a client application includes the format="json" attribute in the <get-configuration> tag.

```
<rpc>
        <get-configuration format="json"/>

    <!-- OR -->

        <get-configuration format="json">
            <!-- tag elements for the configuration elements to return -->
        </get-configuration>
</rpc>
```

When the application requests JSON format, the Junos XML protocol server encloses the JSON data in an <rpc-reply> element.

```
<rpc-reply xmlns:junos="URL">
     <!-- JSON configuration data -->
</rpc-reply>
```

NOTE: Junos OS configuration data emitted in JSON format does not enclose integers in quotation marks.

Include Additional Attributes

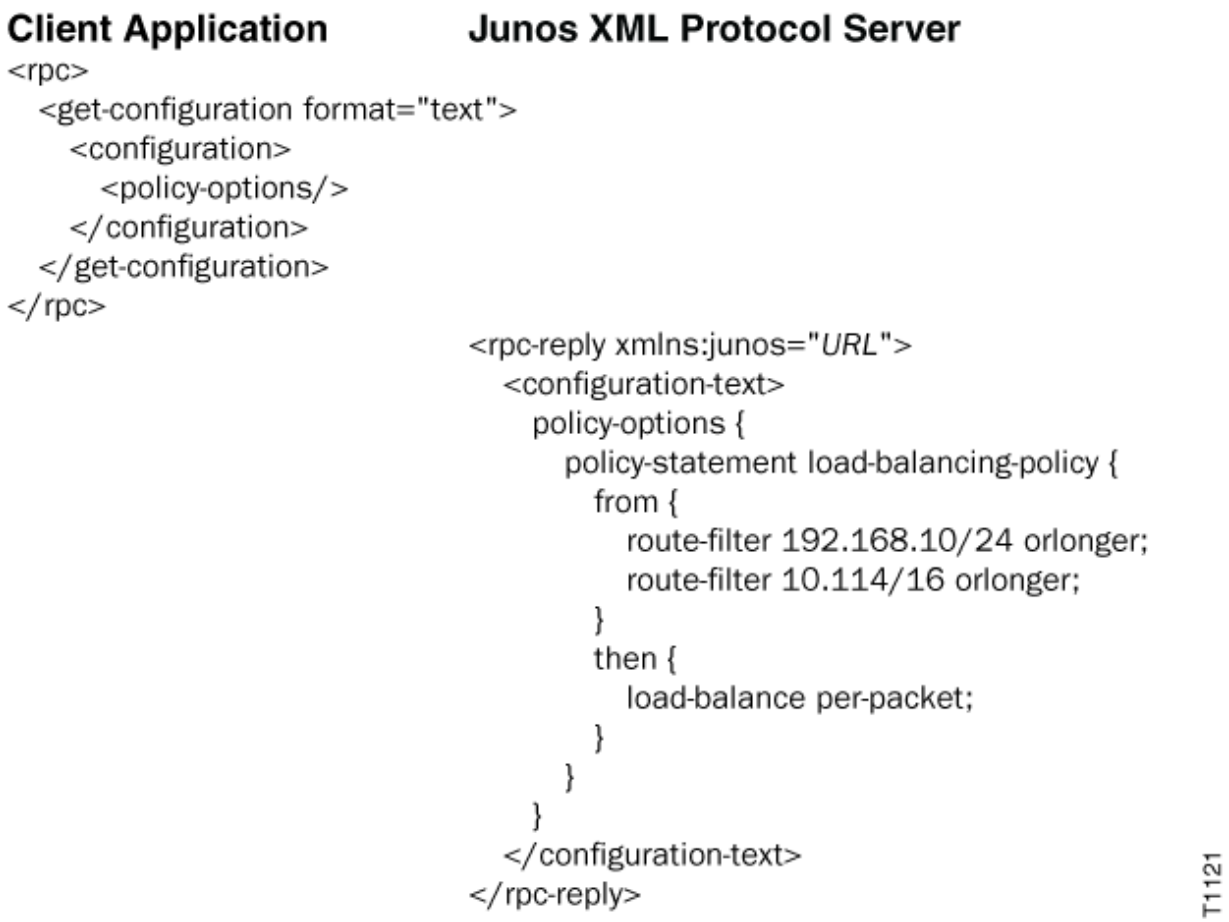
The `format` attribute can be combined with one or more of the following other attributes in the `<get-configuration/>` tag:

- `compare` with the value `compare="rollback"` and with `rollback="0"`. When you compare the candidate configuration to the active configuration (`rollback="0"`), you can display the differences in formats other than text by including the appropriate value for the `format` attribute in the request.
- `commit-scripts` with a value of `commit-scripts="apply"` or `commit-scripts="apply-no-transients"`. The `commit-scripts="view"` attribute returns Junos XML-tagged output by default, even if the `format="text"` attribute is included, since this is the format that is input to commit scripts.
- `database`, which is described in "Specify the Database Source for Configuration Data to Return" on page 377.
- `inherit` and optionally `groups` and `interface-ranges`, which are described in "Specify How to Display Inheritance for Configuration Groups and Interface Ranges Using the Junos XML Protocol" on page 391.

The change and identifier indicators appear only in Junos XML-tagged output and JSON output. Thus, you cannot combine the `format="text"` attribute with the `changed` attribute or include it after requesting an indicator for identifiers.

Example: Request Configuration Data and Specify the Format

The following example requests the `[edit policy-options]` hierarchy level in the candidate configuration as formatted ASCII output.



Specify the Scope of the Configuration Data to Return

A Junos XML protocol client application can use the `<get-configuration>` operation to request configuration data from a Junos device. A client application can request the entire configuration or specific portions of the configuration from the device. To request the complete or partial configuration, a client application encloses the `<get-configuration>` operation in an `<rpc>` element and includes the appropriate child tag elements.

For information about specifying the scope of the configuration information to return, see the following topics:

- "Request the Complete Configuration Using the Junos XML Protocol" on page 414
- "Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol" on page 416
- "Request All Configuration Objects of a Specific Type Using the Junos XML Protocol" on page 419
- "Request a Specific Number of Configuration Objects Using the Junos XML Protocol" on page 420

- "Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol" on page 424
- "Request a Single Configuration Object Using the Junos XML Protocol" on page 427
- "Request Subsets of Configuration Objects Using Regular Expressions" on page 430
- "Request Multiple Configuration Elements Using the Junos XML Protocol" on page 434

RELATED DOCUMENTATION

Understanding the Request Procedure in a Junos XML Protocol Session | 57

<get-configuration> | 123

Request Operational Information Using the Junos XML Protocol | 359

Request Commit-Script-Style XML Configuration Data Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use the `<get-configuration>` operation with the `commit-scripts` attribute to view the configuration in commit-script-style XML.

When you perform a commit operation, each active commit script receives and inspects the post-inheritance candidate configuration in Extensible Markup Language (XML) format. The post-inheritance candidate configuration displays the elements inherited from user-defined groups or interface ranges within the inheriting elements. However, when you view the configuration in the Junos OS CLI using the `show configuration | display xml` command, the device displays the pre-inheritance candidate configuration. In the CLI, you can include additional filters to view the post-inheritance configuration that would be input to a commit script. Additionally, you can view the configuration with commit script changes applied.

Similarly, when a Junos XML protocol client application uses the `<get-configuration>` operation, the server returns the pre-inheritance candidate configuration by default. A client application can also request the configuration as commit-script-style XML data by including the `commit-scripts` attribute. The value of the

attribute determines which view the server returns. Table 19 on page 389 outlines the attribute values, the equivalent CLI command, and a description of the returned data.

Table 19: `commit-scripts` Attribute Values

`commit-scripts` Attribute Value	CLI Command	Description
`commit-scripts="view"`	`show configuration | display commit-scripts view`	View the post-inheritance configuration that would be input to a commit script.
`commit-scripts="apply"`	`show configuration | display commit-scripts`	View the configuration with commit script changes applied, including both non-transient and transient changes.
`commit-scripts="apply-no-transients"`	`show configuration | display commit-scripts no-transients`	View the configuration with commit script changes applied, but include only non-transient changes and exclude transient changes.

The following sections detail how a Junos XML protocol client application requests the configuration in commit-script style XML.

Display the Configuration As Commit Script Input

A Junos XML protocol client application can request the configuration as commit-script-style XML data. To return the post-inheritance configuration that would be input to a commit script, a client application emits the `<rpc>` and `<get-configuration>` elements and includes the `commit-scripts="view"` attribute.

```
<rpc>
    <get-configuration commit-scripts="view"/>

  <!-- OR -->

    <get-configuration commit-scripts="view">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

Display the Configuration with Commit Script Changes Applies

In the Junos OS CLI, you can view the configuration with commit script changes applied to verify that the active commit scripts apply your expected changes. Similarly, a Junos XML protocol client can use the <get-configuration> operation to request the configuration with commit script changes applied.

To view the configuration with both non-transient and transient commit script changes applied, a client application includes the commit-scripts="apply" attribute in the <get-configuration> tag.

```
<rpc>
    <get-configuration commit-scripts="apply"/>

  <!-- OR -->

    <get-configuration commit-scripts="apply">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

To view the configuration with only non-transient commit script changes applied, excluding transient changes, a client application includes the commit-scripts="apply-no-transients" attribute in the <get-configuration> tag.

```
<rpc>
    <get-configuration commit-scripts="apply-no-transients"/>

  <!-- OR -->

    <get-configuration commit-scripts="apply-no-transients">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

Combine the commit-scripts Attribute with Other Attributes

You can combine the commit-scripts attribute with one or more of the following attributes in the <get-configuration> tag:

- changed
- database

- format
- groups
- inherit
- interface-ranges
- junos:key

You do not need to include the changed, groups, or inherit attributes with the commit-scripts="view" attribute. The commit-scripts-style XML view includes the junos:changed="changed" attribute in the XML tags, and it displays the output with inheritance applied. Thus, it displays the tag elements inherited from user-defined groups or interface ranges within the inheriting tag elements, and the XML tags already include the junos:group attribute. To explicitly display the junos:interface-range attribute in the commit-scripts-style view, you must include the interface-ranges="interface-ranges" attribute in the <get-configuration> tag.

When a client application includes the commit-scripts="apply" or commit-scripts="apply-no-transients" attribute, it can request the output as formatted ASCII text, configuration mode commands, or JSON. To specify the format, a client application includes the format attribute and sets it to the required format.

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Request Identifiers for Configuration Elements Using the Junos XML Protocol | 406

Request Change Indicators for Configuration Elements Using the Junos XML Protocol | 411

Specify How to Display Inheritance for Configuration Groups and Interface Ranges Using the Junos XML Protocol

IN THIS SECTION

The following sections explain how to display groups and interface range configurations within their inheriting elements in configuration data that you request through a Junos XML protocol session. The sections also discuss how to view the source group or source interface range for configuration elements that are inherited from a group or interface range.

Understanding Groups and Interface Ranges

The <groups> element corresponds to the [edit groups] configuration hierarchy. It encloses tag elements representing *configuration groups*, each of which contains a set of configuration statements that are appropriate at multiple locations in the hierarchy. You use the apply-groups configuration statement or <apply-groups> tag to insert a configuration group at the appropriate location, achieving the same effect as directly inserting the statements defined in the group. When a configuration group is applied to a configuration hierarchy, the hierarchy is said to *inherit* the group's statements.

In addition to the groups defined at the [edit groups] hierarchy level, Junos OS predefines a group called junos-defaults. This group includes configuration statements judged appropriate for basic operations on the device. By default, CLI commands that display the configuration do not display the statements in the junos-defaults group. Similarly, the Junos XML protocol server output for the <get-configuration> operation does not display this group by default.

The <interface-range> element corresponds to the [edit interfaces interface-range] configuration hierarchy. An interface range is a set of interfaces to which you can apply a common configuration profile. If an interface is a member of an interface range, it inherits the configuration statements set for that range.

Specify How to Display Configuration Groups and Interface Ranges

By default, the Junos XML protocol server displays the element for each user-defined configuration group as a child of the <groups> element, instead of displaying them as children of the elements to which they are applied. Similarly, the server displays the elements for each user-defined interface range as a child of the <interface-range> element, instead of displaying them as children of the elements that are members of the interface range. This display mode parallels the default behavior of the CLI configuration mode show command, which displays [edit groups] and [edit interfaces interface-range] as separate hierarchies in the configuration.

A client application can request that the Junos XML protocol server display elements inherited from user-defined groups or interface ranges within the inheriting elements. To display the inherited elements, a client application includes the `inherit="inherit"` attribute in the `<get-configuration>` tag.

```
<rpc>
    <get-configuration inherit="inherit"/>

     <!-- OR -->

    <get-configuration inherit="inherit">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

To display the inherited elements for user-defined configuration groups and interface ranges and also display the inherited elements from the `junos-defaults` group, a client application includes the `inherit="defaults"` attribute in the `<get-configuration>` tag.

```
<rpc>
    <get-configuration inherit="defaults"/>

     <!-- OR -->

    <get-configuration inherit="defaults">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

When the client includes the `inherit="inherit"` attribute, the output includes the same information as the output from the following CLI configuration mode command. The output does not include configuration elements inherited from the `junos-defaults` group.

```
user@host# show | display inheritance | except ##
```

When the client includes the `inherit="defaults"` attribute, the output includes the same information as the output from the following CLI configuration mode command:

```
user@host# show | display inheritance defaults | except ##
```

The Junos XML protocol server encloses its output in the <rpc-reply> element. Depending on the requested format, the server also includes an additional element. The server emits the <configuration> element for Junos XML-tagged output, the <configuration-text> element for formatted ASCII output, or the <configuration-set> element for configuration mode commands. The server does not enclose JSON data in additional tags.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- Junos XML tag elements representing configuration elements -->
    </configuration>

  <!-- OR -->

    <configuration-text>
        <!-- formatted ASCII configuration statements -->
    </configuration-text>

  <!-- OR -->

    <configuration-set>
        <!-- configuration mode  commands -->
    </configuration-set>

  <!-- OR -->

    <!-- JSON-formatted configuration data -->
</rpc-reply>
```

You can combine the inherit attribute with one or more of the following attributes in the <get-configuration/ tag:

- changed
- database
- format
- groups
- interface-ranges
- junos:key

The application can also include the inherit attribute after requesting an indicator for identifiers (as described in "Request Identifiers for Configuration Elements Using the Junos XML Protocol" on page 406).

Display the Source Group for Inherited Configuration Group Elements

A client application can request that the Junos XML protocol server display the configuration group from which each configuration element is inherited. To display the source group, a client application combines the inherit attribute with the groups="groups" attribute in the <get-configuration> tag.

```
<rpc>
    <get-configuration inherit="(defaults | inherit)" groups="groups"/>

    <!-- OR -->

    <get-configuration inherit="(defaults | inherit)" groups="groups">
        <!-- tag elements indicating  the configuration elements to return -->
    </get-configuration>
</rpc>
```

When you include both the inherit and groups="groups" attributes in the request, the Junos XML protocol server displays each configuration group element within its inheriting element. In addition, the inherited element includes information that indicates the source group. The format determines how the output displays the source group information in the resulting configuration.

If the output is Junos XML format, the server includes the junos:group="*source-group*" attribute in the opening tags of configuration elements that are inherited from configuration groups. The server encloses the data in <rpc-reply> and <configuration> elements.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
    <!-- For each inherited element -->
        <!-- opening-tags-for-parents-of-the-element -->
            <inherited-element junos:group="source-group">
                <inherited-child-of-inherited-element junos:group="source-group">
                    <!-- inherited-children-of-child junos:group="source-group" -->
                </inherited-child-of-inherited-element>
            </inherited-element>
        <!-- closing-tags-for-parents-of-the-element -->
    </configuration>
</rpc-reply>
```

If the output is formatted ASCII text, the server inserts three commented lines with the information about the source group immediately preceding each inherited element. The server encloses the data in <rpc-reply> and <configuration-text> elements.

```
<rpc-reply xmlns:junos="URL">
    <configuration-text>
*/ For each inherited element */
        /* parent levels for the element */
            ##
            ## 'inherited-element' was inherited from group 'source-group'
            ##
            inherited-element {
                ##
                ## 'inherited-child' was inherited from group 'source-group'
                ##
                inherited-child {
                    ... child statements of inherited-child ...
                }
            }
        /* closing braces for parent levels of the element */
    </configuration-text>
</rpc-reply>
```

If the output is in JSON format, the server includes the "junos:group" : "*source-group*" attribute in the attribute list for the inherited element. The server encloses the data in an <rpc-reply> element.

```
<rpc-reply xmlns:junos="URL">
  {
      "configuration" : {
          /* JSON objects for parent levels of the element */
              "inherited-child" : {
                  "@" : {
                      "junos:group" : "source-group"
                  },
                  "inherited-object" : [
                  {
                      "@" : {
                          "junos:group" : "source-group"
                      },
                      "name" : "identifier"
                  }
```

```
                ],
                "@inherited-statement" : {
                    "junos:group" : "source-group"
                }
            }
        /* closing braces for parent levels of the element */
    }
  }
</rpc-reply>
```

When the groups="groups" attribute is combined with the inherit="inherit" attribute, the XML output includes the same information as the output from the following CLI configuration mode command. The output does not include configuration elements inherited from the junos-defaults group:

```
user@host# show | display inheritance | display xml groups
```

When the groups="groups" attribute is combined with the inherit="defaults" attribute, the XML output includes the same information as the output from the following CLI configuration mode command:

```
user@host# show | display inheritance defaults | display xml groups
```

You can combine the inherit and groups attributes with one or more of the following other attributes in the <get-configuration> tag:

- changed
- database
- format
- interface-ranges
- junos:key

The application can also include the inherit and groups attributes after requesting an indicator for identifiers (as described in "Request Identifiers for Configuration Elements Using the Junos XML Protocol" on page 406).

Display the Source Interface Range for Inherited Configuration Elements

A client application can request that the Junos XML protocol server display the interface range from which each configuration element is inherited. To request the source interface range, a client application

combines the `inherit` attribute with the `interface-ranges="interface-ranges"` attribute in the `<get-configuration>` tag.

```
<rpc>
    <get-configuration inherit="inherit" interface-ranges="interface-ranges"/>

  <!-- OR -->

    <get-configuration inherit="inherit" interface-ranges="interface-ranges">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

When you include both the `inherit` and `interface-ranges="interface-ranges"` attributes in the request, the Junos XML protocol server displays each interface range configuration element within its inheriting element. In addition, the inherited element includes information that indicates the source interface range. The format determines how the output displays the source interface range information in the resulting configuration.

If the output is Junos XML format, the server includes the `junos:interface-range="`*source-interface-range*`"` attribute in the opening tags of configuration elements that are inherited from an interface range. The server encloses the data in `<rpc-reply>` and `<configuration>` elements.

```
<rpc-reply xmlns:junos="URL">
   <configuration attributes>
       <interfaces>
           <!-- For each inherited element -->
           <interface junos:interface-range="source-interface-range">
               <inherited-element junos:interface-range="source-interface-range">
                   <inherited-child-of-inherited-element
                           junos:interface-range="source-interface-range">
                      <!-- inherited-children-of-child
                              junos:interface-range="source-interface-range" -->
                   </inherited-child-of-inherited-element>
               </inherited-element>
           </interface>
       </interfaces>
   </configuration>
</rpc-reply>
```

If the output is formatted ASCII text, the server inserts three commented lines with the information about the source interface range immediately preceding each inherited element. The server encloses the data in <rpc-reply> and <configuration-text> elements.

```
<rpc-reply xmlns:junos="URL">
  <configuration-text>
      interfaces {
          <!-- For each inherited element -->
          ##
          ## 'interface-name' was expanded from interface-range 'source-interface-range'
          ##
          interface-name {
              ##
              ## 'inherited-element' was expanded from interface-range 'source-interface-range'
              ##
              inherited-element {
                  inherited-child {
                      ...  child statements of inherited-child ...
                  }
              }
          }
      }
  </configuration-text>
</rpc-reply>
```

If the output is JSON format, the server includes the "junos:interface-range" : "*source-interface-range*" attribute in the attribute list for the inherited element. The server encloses the data in an <rpc-reply> element.

```
<rpc-reply xmlns:junos="URL">
  {
      "configuration" : {
          /* JSON objects for parent levels of the element */
              "inherited-child" : {
                  "@" : {
                      "junos:interface-range" : "source-interface-range"
                  },
                  "inherited-object" : [
                  {
                      "@" : {
                          "junos:interface-range" : "source-interface-range"
```

```
                    },
                    "name" : "identifier"
                }
                ],
                "@inherited-statement" : {
                    "junos:interface-range" : "source-interface-range"
                }
            }
        /* closing braces for parent levels of the element */
    }
  }
</rpc-reply>
```

When you combine the `interface-ranges="interface-ranges"` attribute with the `inherit="inherit"` attribute, the XML output includes the same information as the output from the following CLI configuration mode command:

```
user@host# show | display inheritance | display xml interface-ranges
```

When you combine the `interface-ranges="interface-ranges"` attribute with the `inherit="defaults"` attribute, the XML output includes the same information as the output from the following CLI configuration mode command:

```
user@host# show | display inheritance defaults | display xml interface-ranges
```

You can combine the `inherit` and `interface-ranges` attributes with one or more of the following other attributes in the `<get-configuration/` tag:

- `changed`
- `database`
- `format`
- `groups`
- `junos:key`

The application can also include the `inherit` and `interface-ranges` attributes after requesting an indicator for identifiers.

Summary of Attributes for Configuration Group and Interface Range Inheritance

Table 20 on page 401 summarizes the <get-configuration> attributes for inheritance, the impact on the configuration data, and the CLI command that produces equivalent output.

Table 20: <get-configuration> Attributes for Inheritance

Attribute	Additional Attribute	Description	CLI Command Equivalent
–	–	Display groups and interface ranges as separate elements in the output instead of displaying them within their inheriting elements.	show
inherit="inherit"	–	Display user-defined groups and interface ranges within the inheriting elements.	show | display inheritance | except ##
	groups="groups"	Display user-defined groups and interface ranges within the inheriting elements. Display the source group for each inherited configuration group.	show | display inheritance | display xml groups
	interface-ranges="interface-ranges"	Display user-defined groups and interface ranges within the inheriting elements. Display the source interface range for each inherited interface range.	show | display inheritance | display xml interface-ranges
inherit="defaults"	–	Display user-defined groups and interface ranges and the junos-defaults group within the inheriting elements.	show | display inheritance defaults | except ##

Table 20: <get-configuration> Attributes for Inheritance *(Continued)*

Attribute	Additional Attribute	Description	CLI Command Equivalent
	`groups="groups"`	Display user-defined groups and interface ranges and the `junos-defaults` group within the inheriting elements. Display the source group for each inherited configuration group.	`show | display inheritance defaults | display xml groups`
	`interface-ranges="interface-ranges"`	Display user-defined groups and interface ranges and the `junos-defaults` group within the inheriting elements. Display the source interface range for each inherited interface range.	`show | display inheritance defaults | display xml interface-ranges`

Examples: Specify the Output Format for Configuration Groups

The following sample configuration hierarchy defines a configuration group called `interface-group`. The `apply-groups` statement applies the statements in the group at the `[edit interfaces]` hierarchy level.

```
[edit]
groups {
    interface-group {
        interfaces {
            so-1/1/1 {
                encapsulation ppp;
            }
        }
    }
}
apply-groups interface-group;
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                address 192.168.4.207/24;
            }
```

```
        }
    }
}
```

When you execute the <get-configuration/> operation but omit inherit attribute, the output includes the <groups> and <apply-groups> elements as separate items. The <groups> element encloses the elements defined in the interface-group configuration group. The placement of the <apply-groups> element directly above the <interfaces> element indicates that the [edit interfaces] hierarchy inherits the statements defined in the interface-group configuration group.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration/>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <groups>
      <name>interface-group</name>
      <interfaces>
        <interface>
          <name>so-1/1/1</name>
          <encapsulation>ppp</encapsulation>
        </interface>
      </interfaces>
    </groups>
    <apply-groups>interface-group</apply-groups>
    <interfaces>
      <interface>
        <name>fxp0</name>
        <unit>
          <name>0</name>
          <family>
            <inet>
              <address>
                <name>192.168.4.207/24</name>
              </address>
            </inet>
          </family>
        </unit>
      </interface>
    </interfaces>
  </configuration>
</rpc-reply>
```

T1188

When you execute the `<get-configuration/>` operation and include the `inherit` attribute, the `<interfaces>` element encloses the elements defined in the `interface-group` configuration group. The output does not display the `<groups>` and `<apply-groups>` elements separately.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration inherit="inherit"/>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <interfaces>
      <interface>
        <name>fxp0</name>
        <unit>
          <name>0</name>
          <family>
            <inet>
              <address>
                <name>192.168.4.207/24</name>
              </address>
            </inet>
          </family>
        </unit>
      </interface>
      <interface>
        <name>so-1/1/1</name>
        <encapsulation>ppp</encapsulation>
      </interface>
    </interfaces>
  </configuration>
</rpc-reply>
```

T1189

The following example executes the `<get-configuration/>` operation and combines the `groups="groups"` attribute with the `inherit` attribute. In the output, the `<interfaces>` element encloses the elements defined in the `interface-group` configuration group. The output also provides information about the source group for the inherited elements. The `junos:group="interface-group"` attribute indicates the source group.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration inherit="inherit" groups="groups"/>
</rpc>
                        <rpc-reply xmlns:junos="URL">
                          <configuration junos:changed-seconds="seconds" \
                              junos:changed-localtime="timestamp">
                            <interfaces>
                              <interface>
                                <name>fxp0</name>
                                <unit>
                                  <name>0</name>
                                  <family>
                                    <inet>
                                      <address>
                                        <name>192.168.4.207/24</name>
                                      </address>
                                    </inet>
                                  </family>
                                </unit>
                              </interface>
                              <interface junos:group="interface-group">
                                <name junos:group="interface-group">so-1/1/1</name>
                                <encapsulation junos:group="interface-group">ppp</encapsulation>
                              </interface>
                            </interfaces>
                          </configuration>
                        </rpc-reply>
```

T1190

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Request Identifiers for Configuration Elements Using the Junos XML Protocol

SUMMARY

Junos XML protocol client applications can request configuration data that indicates which elements are identifiers for configuration objects.

A Junos XML protocol client application can request the identifiers for configuration objects on devices running Junos and devices running Junos OS Evolved. To request that the server indicate whether a child configuration element is an identifier for its parent element, a client application can use one of the following methods:

- Include the `junos:key="key"` attribute in the opening `<junoscript>` tag for the Junos XML protocol session.
- Include the `junos:key="key"` attribute or the `key="key"` attribute in the `<get-configuration>` request tag.

```
<junoscript version="version" junos:key="key">

    <!-- OR -->

<rpc>
   <get-configuration (junos:key | key)="key">
       <!-- tag elements for the configuration elements to return -->
   </get-configuration>
</rpc>
```

For more information about the `<junoscript>` tag, see "Start a Junos XML Protocol Session" on page 75.

When the identifier indicator is requested, the Junos XML protocol server includes the `junos:key="key"` attribute in the opening tag for each identifier. As always, the Junos XML protocol server encloses its response in `<rpc-reply>` and `<configuration>` tag elements. In the following example, the identifier tag element is called `<name>`:

```
<rpc-reply xmlns:junos="URL">
        <configuration attributes>
            <!-- opening tag for each parent of the object -->

        <!-- For each configuration object with an identifier -->
            <object>
                <name junos:key="key">identifier</name>
                <!-- additional children of object -->
                </object>
            <!-- closing tag for each parent of the object -->

    </configuration>
</rpc-reply>
```

If the requested output format is JSON, the Junos XML protocol server adds a metadata object that includes `"junos:key" : "key"` to indicate the identifier. If the object uses `name` for the identifier, a metadata object "@" is added as a new member of the object. If the object uses an identifier other than `name`, the metadata object is added as a sibling name/value pair that uses "@" concatenated with the identifier name. The response is enclosed in `<configuration-json>` and `<rpc-reply>` tag elements.

```
<rpc-reply xmlns:junos="URL">
  <configuration-json>
  {
      "configuration" : {
          /* JSON objects for parent levels of the element */
              "object" : [
              {
                  "@" : {
                      "junos:key" : "key"
                  },
                  "name" : "identifier",
                  "identifier-name" : "identifier-value",
                  "@identifier-name" : {
                      "junos:key" : "key"
                  },
                  /* additional data and child objects */  # if any
              }
              ]
          /* closing braces for parent levels of the element */
      }
  }
  </configuration-json>
</rpc-reply>
```

In the following output, the combination of `name` and `next-hop` uniquely identify the static route:

```
{
    "configuration" : {
        "routing-options" : {
            "static" : {
                "route" : [
                {
                    "@" : {
                        "junos:key" : "key"
                    },
```

```
                    "name" : "172.16.0.0/12",
                    "next-hop" : ["198.51.100.1"],
                    "@next-hop" : {
                        "junos:key" : "key"
                    },
                    "retain" : [null],
                    "no-readvertise" : [null]
                }
                ]
            }
        }
    }
}
```

The client application can include one or more of the following attributes in the <get-configuration> tag when it includes the junos:key attribute in either the <junoscript> tag or the <get-configuration> tag:

- changed
- commit-scripts
- database
- inherit and optionally groups and interface-ranges

The junos:key="key" attribute appears only in Junos XML output (the default) and JSON output. Thus, you cannot combine the junos:key="key" attribute with the format="text" attribute or with the compare attribute, which produces only text output.

The following example requests indicators for identifiers for elements in the `[edit interfaces]` hierarchy level of the candidate configuration. The output is truncated for brevity.

Client Application

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" \
            junos:key="key" \
            release="JUNOS-release">

<rpc>
  <get-configuration>
    <configuration>
      <interfaces/>
    </configuration>
  </get-configuration>
</rpc>
```

Junos XML Protocol Server

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" hostname="router1" \
            os="JUNOS" release="JUNOS-release" \
            xmlns="URL"xmlns:junos="URL" \
            xmlns:xnm="URL">

<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <interfaces>
      <!- - tag elements for other interfaces - ->
      <interface>
        <name junos:key="key">lo0</name>
        <unit>
          <name junos:key="key">0</name>
          <family>
            <inet>
              <address>
                <name junos:key="key">127.0.0.1/32</name>
              </address>
            </inet>
          </family>
        </unit>
      </interface>
      <!- - tag elements for other interfaces - ->
    </interfaces>
  </configuration>
</rpc-reply>
```

T1187

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Specify the Output Format for Configuration Data to Return | 382

Request Change Indicators for Configuration Elements Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use the <get-configuration> operation with the changed="changed" attribute to return configuration data that includes indicators for changed configuration elements.

A Junos XML protocol client application can use the <get-configuration> operation to request configuration data from a device running Junos OS or a device running Junos OS Evolved. A client application can request configuration data that includes indicators for configuration elements that have changed since the last commit operation. To request change indicators, the client application includes the changed="changed" attribute in the <get-configuration> tag.

```
<rpc>
    <get-configuration changed="changed"/>

  <!-- OR -->

    <get-configuration changed="changed">
        <!-- tag elements for the configuration elements to return -->
    </get-configuration>
</rpc>
```

The returned configuration includes the junos:changed="changed" attribute for changed elements. The device determines which elements to mark as changed depending on the configuration source. You specify the source by including the database attribute in the <get-configuration> tag. The default is candidate, which retrieves the candidate configuration. The options include:

- database="candidate"—(Default) The device compares the candidate configuration to the active configuration. The configuration includes the junos:changed="changed" attribute for elements that were added to the candidate configuration after the last commit operation.
- database="committed"—The device compares the active configuration to the first rollback configuration. The configuration includes the junos:changed="changed" attribute for elements that were added to the active configuration by the most recent commit.

The server indicates which elements have changed by including the junos:changed="changed" attribute in the opening tag of every parent tag element in the path to the changed configuration element. If the changed configuration element is a single (empty) tag, the tag displays the junos:changed="changed" attribute. If the changed element is a container element, the configuration displays the junos:changed="changed" attribute in the opening container tag and also in the opening tag for each child tag element enclosed in the container element.

The Junos XML protocol server encloses its response in <rpc-reply> and <configuration> elements.

```
<rpc-reply xmlns:junos="URL">
    <configuration standard-attributes junos:changed="changed">
        <!-- opening-tag-for-each-parent-level junos:changed="changed" -->

      <!-- For each changed element, EITHER -->
        <element junos:changed="changed"/>

      <!-- OR -->

        <element junos:changed="changed">
            <first-child-of-element junos:changed="changed">
            <second-child-of-element junos:changed="changed">
            <!-- additional children of element -->
        </element>

        <!-- closing-tag-for-each-parent-level -->
    </configuration>
</rpc-reply>
```

If the requested output format is JSON, the server includes the "junos:changed" : "changed" attribute in the attribute lists for the same elements as described previously.

NOTE: When a commit operation succeeds, the Junos XML protocol server removes the junos:changed="changed" attribute from all tag elements. However, if the commit generates warnings, the attribute remains. In this case, the junos:changed="changed" attribute appears on tag elements that changed before the commit as well as on tag elements that changed after the commit.

An example of a commit-time warning is the message explaining that a configuration element does not apply until you reboot the device. The Junos XML protocol server returns the warning in an <xnm:error> element when it confirms the success of the commit operation.

> To remove the junos:changed="changed" attribute from elements that changed before the commit, the client application must take the necessary actions to eliminate the cause of the warning, and then commit the configuration again.

You can combine the changed attribute with one or more of the following attributes in the <get-configuration> tag:

- database
- inherit and optionally groups and interface-ranges
- junos:key

The junos:changed="changed" attribute appears only in Junos XML output (the default) and JSON output. Thus, you cannot combine the changed attribute with the format="text" attribute or with the compare attribute, which produces only text output.

When you include the commit-scripts="view" attribute, the junos:changed="changed" attribute is automatically included in the output. Thus, you do not need to explicitly include the changed="changed" attribute in the <get-configuration> request.

The following example requests change indicators for configuration elements at the [edit system syslog] hierarchy level in the candidate configuration. The output indicates that a log file called **interactive-commands** has been configured since the last commit.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration changed="changed">
    <configuration>
      <system>
        <syslog/>
      </system>
    </configuration>
  </get-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp" junos:changed="changed">
    <system junos:changed="changed">
      <syslog junos:changed="changed">
        <file>
          <name>messages</name>
          <contents>
            <name>any</name>
            <info/>
          </contents>
        </file>
        <file junos:changed="changed">
          <name junos:changed="changed">interactive-commands</name>
          <contents>
            <name junos:changed="changed">interactive-commands</name>
            <notice junos:changed="changed"/>
          </contents>
        </file>
      </syslog>
    </system>
  </configuration>
</rpc-reply>
```

T1186

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Request Commit-Script-Style XML Configuration Data Using the Junos XML Protocol | 388

Request the Complete Configuration Using the Junos XML Protocol

A Junos XML protocol client application can request the complete configuration from a device running Junos OS or a device running Junos OS Evolved. To request the entire candidate configuration or the

complete configuration in an open instance of the ephemeral configuration database, a client application emits the <rpc> and <get-configuration/> elements.

```
<rpc>
    <get-configuration/>
</rpc>
```

NOTE: If a client application issues the Junos XML protocol <open-configuration> operation to open a specific configuration database before executing the <get-configuration> operation, the server returns the configuration data from the open configuration database. Otherwise, the server returns the configuration data from the candidate configuration, unless the active configuration is explicitly requested by including the database="committed" attribute.

When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested configuration in <configuration> and <rpc-reply> tag elements. For information about the attributes in the opening <configuration> tag, see "Specify the Database Source for Configuration Data to Return" on page 377.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- Junos XML tag elements for all configuration elements -->
    </configuration>
</rpc-reply>
```

To specify the source of the output (candidate or active configuration) and request special formatting of the output (for example, formatted ASCII or JSON or an indicator for identifiers), the application can include attributes in the <get-configuration/> tag, the opening <junoscript> tag, or both. For more information, see "Specify the Database Source for Configuration Data to Return" on page 377 and "Specify the Output Format for Configuration Data to Return" on page 382.

The following example shows how to request the complete candidate configuration tagged with Junos XML tag elements (the default). In actual output, the *JUNOS-version* variable is replaced by a value such as 20.4R1 for the initial version of Junos OS Release 20.4.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration/>
</rpc>
```

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

T1191

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Retrieve a Previous (Rollback) Configuration Using the Junos XML Protocol | 435
- Retrieve the Rescue Configuration Using the Junos XML Protocol | 443

Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol

In a Junos XML protocol 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 `<get-configuration>` 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 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-configuration>
        <configuration>
            <!-- opening tags for each parent of the level -->
                <requested-level/>
```

```
            <!-- closing tags for each parent of the level -->
        </configuration>
    </get-configuration>
</rpc>
```

When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested section of the configuration in <configuration> and <rpc-reply> tag elements. For information about the attributes in the opening <configuration> tag, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- opening tags for each parent of the level -->
            <hierarchy-level>
                <!-- child tag elements of the level -->
            </hierarchy-level>
        <!-- closing tags for each parent of the level -->
    </configuration>
</rpc-reply>
```

To specify the source of the output (candidate or active configuration) and request special formatting of the output (for example, formatted ASCII or JSON or an indicator for identifiers), the application can include attributes in the opening <get-configuration> tag, its opening <junoscript> tag, or both. For more information, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377 and "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382.

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same <get-configuration> tag element. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

The following example shows how to request the contents of the [edit system login] hierarchy level in the candidate configuration. The output is tagged with Junos XML tag elements, which is the default.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration>
    <configuration>
      <system>
        <login/>
      </system>
    </configuration>
  </get-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <system>
      <login>
        <user>
          <name>barbara</name>
          <full-name>Barbara Anderson</full-name>
          <!- - other child tags for this user - ->
        </user>
        <!- - other children of <login> - ->
      </login>
    </system>
  </configuration>
</rpc-reply>
```

T1192

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Request All Configuration Objects of a Specific Type Using the Junos XML Protocol | 419
- Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420
- Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol | 424

Request All Configuration Objects of a Specific Type Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, to request complete information about all configuration objects of a specified type in a hierarchy level, a client application emits a <get-configuration> 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. An empty tag represents the requested object type. The entire request is enclosed in an <rpc> tag element.

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object type -->
                <object-type/>
            <!-- closing tags for each parent of the object type -->
        </configuration>
    </get-configuration>
</rpc>
```

This type of request is useful when the object's parent hierarchy level has child objects of multiple types and the application is requesting just one of the types. If the requested object is the only possible child type, then this type of request yields the same output as a request for the complete parent hierarchy (described in "Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol" on page 416).

When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested objects in <configuration> and <rpc-reply> tag elements. For information about the attributes in the opening <configuration> tag, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

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

To specify the source of the output (candidate or active configuration) and request special formatting of the output (for example, formatted ASCII or JSON or an indicator for identifiers), the application can include attributes in the opening <get-configuration> tag, its opening <junoscript> tag, or both. For more information, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377 and "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382.

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same <get-configuration> tag element. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol | 416
- Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420
- Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol | 424

Request a Specific Number of Configuration Objects Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, to request information about a specific number of configuration objects of a specific type, a client application emits the <get-configuration> tag element and encloses the tag elements that represent 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. An empty tag represents the requested object type, and the tag includes the following attributes:

- count specifies the number of objects to return
- start specifies the index number of the first object to return (1 for the first object, 2 for the second, and so on)

If the application is requesting only the first object in the hierarchy, it includes the `count="1"` attribute and omits the `start` attribute. The application encloses the entire request in an `<rpc>` tag element.

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object -->
                <object-type  count="count" start="index"/>
            <!-- closing tags for each parent of the object -->
        </configuration>
    </get-configuration>
</rpc>
```

NOTE: The `count` and `start` attributes are not supported when requesting configuration data in JSON format.

The Junos XML protocol server returns the requested objects starting with the object specified by the `start` attribute and running consecutively. When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested objects in `<configuration>` and `<rpc-reply>` tag elements, starting with the object specified by the `start` attribute and running consecutively.

For each object, the server includes two attributes:

- `junos:position`, to specify the object's numerical index
- `junos:total`, to report the total number of such objects that exist in the hierarchy

In the following example Junos XML output, 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 in a Junos XML Protocol Session" on page 377.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- opening tags for each parent of the object type -->
            <first-object junos:position="index1" junos:total="total">
                <name>identifier-for-first-object</name>
                <!-- other child tag elements of the first object -->
            </first-object>
            <second-object junos:position="index2" junos:total="total">
                <name>identifier-for-second-object</name>
                <!-- other child tag elements of the second object -->
```

```
            </second-object>
            <!-- additional objects -->
        <!-- closing tags for each parent of the object type -->
    </configuration>
</rpc-reply>
```

The `junos:position` and `junos:total` attributes do not appear if the client requests formatted ASCII output by including the `format="text"` attribute in the `<get-configuration>` tag element (as described in "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382).

To specify the source of the output (candidate or active configuration), the application can include attributes in the opening `<get-configuration>` tag, its opening `<junoscript>` tag, or both. For more information, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same `<get-configuration>` tag element. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

The following example shows how to request the third and fourth Junos user accounts at the `[edit system login]` hierarchy level. The output is from the candidate configuration and is tagged with Junos XML tag elements (the default).

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration>
    <configuration>
      <system>
        <login>
          <user count="2" start="3"/>
        </login>
      </system>
    </configuration>
  </get-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <system>
      <login>
        <user junos:position="3" junos:total="22">
          <name>barbara</name>
          <uid>1423</uid>
          <class>operator</class>
        </user>
        <user junos:position="4" junos:total="22">
          <name>carlo</name>
          <uid>1426</uid>
          <class>operator</class>
        </user>
      </login>
    </system>
  </configuration>
</rpc-reply>
```

T1193

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol | 416
- Request All Configuration Objects of a Specific Type Using the Junos XML Protocol | 419
- Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol | 424

Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol

In a Junos XML protocol 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 <get-configuration> 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. An empty tag represents the requested object type, and the recurse="false" attribute is included. The entire request is enclosed in an <rpc> tag element.

To request the identifier for all objects of a specified type, the client application includes only the recurse="false" attribute:

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object type -->
                <object-type recurse="false"/>
            <!-- closing tags for each parent of the object type -->
        </configuration>
    </get-configuration>
</rpc>
```

To request the identifier for a specified number of objects, the client application combines the recurse="false" attribute with the count and start attributes discussed in "Requesting a Specific Number of Configuration Objects Using the Junos XML Protocol" on page 420:

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the object type -->
                <object-type  recurse="false" count="count" start="index"/>
            <!-- closing tags for each parent of the object type -->
        </configuration>
    </get-configuration>
</rpc>
```

When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested objects in <configuration> and <rpc-reply> tag elements. If the application has requested a specified number of objects, the junos:position and junos:total attributes are included in the

opening tag for each object, as described in "Requesting a Specific Number of Configuration Objects Using the Junos XML Protocol" on page 420.

In the following example output, 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 in a Junos XML Protocol Session" on page 377.)

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- opening tags for each parent of the object type -->
            <first-object [junos:position="index1" junos:total="total"]>
                <name>identifier-for-first-object</name>
            </first-object>
            <second-object [junos:position="index2" junos:total="total"]>
                <name>identifier-for-second-object</name>
            </second-object>
            <!-- additional instances of the object -->
        <!-- closing tags for each parent of the object type -->
    </configuration>
</rpc-reply>
```

The junos:position and junos:total attributes do not appear if the client requests formatted ASCII output by including the format="text" attribute or if the client requests JSON-formatted output by including the format="json" attribute in the <get-configuration> tag element (as described in "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382).

To specify the source of the output (candidate or active configuration), the application can include attributes in the opening <get-configuration> tag, its opening <junoscript> tag, or both. For more information, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same <get-configuration> tag element. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

The following example shows how to request the identifier for each interface configured at the [edit interfaces] hierarchy level. The output is from the candidate configuration and is tagged with Junos XML tag elements (the default).

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration>
    <configuration>
      <interfaces>
        <interface recurse="false"/>
      </interfaces>
    </configuration>
  </get-configuration>
</rpc>
```

```
<rpc-reply xmlns:junos="URL">
  <configuration junos:changed-seconds="seconds" \
      junos:changed-localtime="timestamp">
    <interfaces>
      <interface>
          <name>fe-0/0/0</name>
      </interface>
      <interface>
          <name>fxp0</name>
      </interface>
      <interface>
          <name>lo0</name>
      </interface>
    </interfaces>
  </configuration>
</rpc-reply>
```

T1194

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol | 416
- Request All Configuration Objects of a Specific Type Using the Junos XML Protocol | 419
- Request a Specific Number of Configuration Objects Using the Junos XML Protocol | 420

Request a Single Configuration Object Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, to request complete information about a single configuration object, a client application emits the <get-configuration> 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-configuration>
        <configuration>
            <!-- opening tags for each parent of the object -->
                <object>
                    <name>identifier</name>
                </object>
            <!-- closing tags for each parent of the object -->
        </configuration>
    </get-configuration>
</rpc>
```

When the client application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested object in <configuration> and <rpc-reply> tag elements. For information about the attributes in the opening <configuration> tag, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- opening tags for each parent of the object -->
            <object>
                <!-- child tag elements of the object -->
            </object>
        <!-- closing tags for each parent of the object -->
```

```
    </configuration>
</rpc-reply>
```

To specify the source of the output (candidate or active configuration) and request special formatting of the output (for example, formatted ASCII or JSON or an indicator for identifiers), the application can include attributes in the opening <get-configuration> tag, its opening <junoscript> tag, or both. For more information, see "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377 and "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382.

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same <get-configuration> tag element. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

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. To specify the desired object, the client application emits the <name>local</name> identifier tag element as the innermost tag element. The output is from the candidate configuration and is tagged with Junos XML tag elements (the default).

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-configuration>
    <configuration>
      <routing-options>
        <multicast>
          <scope>
            <name>local</name>
          </scope>
        </multicast>
      </routing-options>
    </configuration>
  </get-configuration>
</rpc>
```

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

T1195

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Request a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol | 416
- Request All Configuration Objects of a Specific Type Using the Junos XML Protocol | 419
- Request Subsets of Configuration Objects Using Regular Expressions | 430
- Request Multiple Configuration Elements Using the Junos XML Protocol | 434

Request Subsets of Configuration Objects Using Regular Expressions

A Junos XML protocol client application can request information for configuration object instances that include a specified set of characters in their identifiers. To filter the objects by identifier, a client application includes the `matching` attribute with a regular expression that matches on the identifiers. For example, the application can specify a regular expression that matches on the characters "ge-" to request information about the Gigabit Ethernet interfaces at the `[edit interfaces]` hierarchy level.

The `matching` attribute enables the application to represent the objects to return in a form similar to the XML Path Language (XPath) representation. For example, the following XPath:

```
configuration/system/radius-server/name
```

is equivalent to the following tagged representation:

```
<configuration>
    <system>
        <radius-server>
            <name/>
        </radius-server>
    </system>
</configuration>
```

For more information about XPath, see *XML Path Language (XPath) Version 1.0*.

To filter by identifier, the application includes the `matching` attribute in the empty tag that represents a parent level for the object type. As with all requests for configuration information, the client emits a `<get-configuration>` 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 level at which the `matching` attribute is included.

```
<rpc>
    <get-configuration>
        <configuration>
            <!-- opening tags for each parent of the level -->
                <level matching="matching-expression"/>
            <!-- closing tags for each parent of the level -->
        </configuration>
    </get-configuration>
</rpc>
```

The `matching` attribute is case-insensitive. In the value for the `matching` attribute, each level in the XPath-like representation can be either a full level name or a regular expression that matches the identifier name of one or more instances of an object type. For example:

```
object-type[name='regular-expression']
```

The regular expression uses the notation defined in POSIX Standard 1003.2 for extended (modern) UNIX regular expressions. Table 21 on page 431 specifies which character or characters are matched by some of the regular expression operators that you can use in the expression. In the descriptions, the term *term* refers to either a single alphanumeric character or a set of characters enclosed in square brackets, parentheses, or braces.

Table 21: Regular Expression Operators for the matching Attribute

Operator	Matches
. (period)	One instance of any character.
* (asterisk)	Zero or more instances of the immediately preceding term.
+ (plus sign)	One or more instances of the immediately preceding term.
? (question mark)	Zero or one instance of the immediately preceding term.
| (pipe)	One of the terms that appear on either side of the pipe operator.
^ (caret)	The start of a line, when the caret appears outside square brackets. One instance of any character that does not follow it within square brackets, when the caret is the first character inside square brackets.
$ (dollar sign)	The end of a line.
[ ] (paired square brackets)	One instance of one of the enclosed alphanumeric characters. To indicate a range of characters, use a hyphen (-) to separate the beginning and ending characters of the range. For example, `[a-z0-9]` matches any letter or number.

Table 21: Regular Expression Operators for the matching Attribute *(Continued)*

Operator	Matches
( ) (paired parentheses)	One instance of the evaluated value of the enclosed term. Parentheses are used to indicate the order of evaluation in the regular expression.

When the application requests Junos XML-tagged output (the default), the Junos XML protocol server returns the requested object in `<rpc-reply>` and `<configuration>` elements.

```
<rpc-reply xmlns:junos="URL">
    <configuration attributes>
        <!-- opening tags for each parent of the parent level -->
            <parent-level>
                <first-matching-object>
                    <!-- child tag elements for the first object -->
                </first-matching-object>
                <second-matching-object>
                    <!-- child tag elements for the second object -->
                </second-matching-object>
                <!-- additional instances of the object -->
            </parent-level>
        <!-- closing tags for each parent of the object type -->
    </configuration>
</rpc-reply>
```

The application can combine one or more of the `count`, `start`, and `recurse` attributes along with the `matching` attribute. These attributes enable you to limit the set of possible matches to a specific range of objects, to request only identifiers, or both. For more information about these attributes, see:

- "Request a Specific Number of Configuration Objects Using the Junos XML Protocol" on page 420
- "Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol" on page 424

To specify the source of the output (candidate or active configuration) and request special formatting of the output (for example, formatted ASCII or JSON or an indicator for identifiers), the application can include attributes in the opening `<get-configuration>` tag, its opening `<junoscript>` tag, or both. For more information, see:

- "Specifying the Source for Configuration Information Requests in a Junos XML Protocol Session" on page 377

- "Specifying the Output Format for Configuration Data in a Junos XML Protocol Session" on page 382

The same <get-configuration> operation can also request additional configuration elements of the same or other types in by including the appropriate tag elements. For more information, see "Requesting Multiple Configuration Elements Using the Junos XML Protocol" on page 434.

The following example shows how to request just the identifier for the first two Gigabit Ethernet interfaces configured at the [edit interfaces] hierarchy level:

```
<rpc>
  <get-configuration>
    <configuration>
      <interfaces matching="interface[name='ge-.*']" count="2" recurse="false"/>
    </configuration>
  </get-configuration>
</rpc>
```

The Junos XML protocol server returns the identifiers for the requested objects.

```
<rpc-reply xmlns:junos="URL">
<configuration junos:changed-seconds="1758308076" junos:changed-localtime="2025-09-19 11:54:36
PDT">
    <interfaces>
        <interface junos:position="1" junos:total="2">
            <name>ge-0/0/0</name>
        </interface>
        <interface junos:position="2" junos:total="2">
            <name>ge-0/0/1</name>
        </interface>
    </interfaces>
</configuration>
</rpc-reply>
```

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Specify the Scope of the Configuration Data to Return | 387

Request the Complete Configuration Using the Junos XML Protocol | 414

Request a Single Configuration Object Using the Junos XML Protocol | 427

Request Multiple Configuration Elements Using the Junos XML Protocol

In a Junos XML protocol 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-configuration> tag element. The request includes only one <configuration> tag element (the Junos XML protocol server returns an error if there is more than one).

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 parent tag element -->
<rpc>
    <get-configuration>
        <configuration>
            <system>
                <services/>
                <radius-server>
                    <name/>
                </radius-server>
            </system>
        </configuration>
    </get-configuration>
</rpc>

<!-- separate parent tag element for each request -->
<rpc>
    <get-configuration>
        <configuration>
            <system>
                <services/>
            </system>
            <system>
                <radius-server>
                    <name/>
                </radius-server>
            </system>
        </configuration>
    </get-configuration>
</rpc>
```

The client can combine requests for any of the types of information discussed in the following sections:

- "Requesting a Configuration Hierarchy Level or Container Object Without an Identifier Using the Junos XML Protocol" on page 416
- "Requesting All Configuration Objects of a Specific Type Using the Junos XML Protocol" on page 419
- "Requesting a Specific Number of Configuration Objects Using the Junos XML Protocol" on page 420
- "Requesting Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol" on page 424
- "Requesting a Single Configuration Object Using the Junos XML Protocol" on page 427
- "Requesting Subsets of Configuration Objects Using Regular Expressions" on page 430

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Specify the Scope of the Configuration Data to Return | 387

Request the Complete Configuration Using the Junos XML Protocol | 414

Request Identifiers for Configuration Objects of a Specific Type Using the Junos XML Protocol | 424

Request a Single Configuration Object Using the Junos XML Protocol | 427

Request Subsets of Configuration Objects Using Regular Expressions | 430

Retrieve a Previous (Rollback) Configuration Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can retrieve a previously committed configuration by referencing its rollback index or its configuration revision identifier.

IN THIS SECTION

Retrieve a Configuration Using the Configuration Revision Identifier | 441

A Junos XML protocol client application can request a previously committed (rollback) configuration from a device running Junos OS or a device running Junos OS Evolved. The client can retrieve the configuration by referencing the configuration's rollback index or its configuration revision identifier.

Understanding the Rollback Index and Configuration Revision Identifier

Junos OS and Junos OS Evolved store a copy of the most recently committed configuration and up to 49 previous configurations, depending on the platform. When you successfully commit a configuration, Junos OS assigns that configuration a unique configuration revision identifier. The configuration is also associated with a rollback index, where the most recently committed configuration has rollback index 0. Whereas the rollback index for a previously committed configuration increments with each commit, the configuration revision ID remains static for the same configuration. When you request a previously committed configuration, you can reference the configuration by its current rollback index or its static configuration revision ID.

A client application can view a device's commit history to see the configuration revision identifier and the rollback index for committed configurations. To view the commit history and include the configuration revision IDs, a client application executes the `<get-commit-information>` RPC with the `<include-configuration-revision/>` child element.

```
<rpc>
  <get-commit-information>
    <include-configuration-revision/>
  </get-commit-information>
</rpc>
```

The server returns XML output equivalent to the `show system commit include-configuration-revision` operational mode command.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.4R1/junos">
<commit-information>
<commit-history>
<sequence-number>0</sequence-number>
<user>admin</user>
<client>netconf</client>
<date-time junos:seconds="1605236880">2020-11-12 19:08:00 PST</date-time>
```

```
<configuration-revision>re0-1605236878-333</configuration-revision>
</commit-history>
<commit-history>
<sequence-number>1</sequence-number>
<user>user1</user>
<client>netconf</client>
<date-time junos:seconds="1605228068">2020-11-12 16:41:08 PST</date-time>
<configuration-revision>re0-1605228066-332</configuration-revision>
</commit-history>
</commit-information>
<commit-history>
<sequence-number>2</sequence-number>
<user>admin</user>
<client>cli</client>
<date-time junos:seconds="1605226205">2020-11-12 16:10:05 PST</date-time>
<configuration-revision>re0-1605226203-331</configuration-revision>
</commit-history>
...
</rpc-reply>
```

If you already know the configuration revision ID for a particular configuration, you can determine the mapping between the ID and the current rollback index. Remember that the configuration revision ID for a given committed configuration is static whereas the rollback index increments with each commit.

To determine the rollback number corresponding to a specific configuration revision ID, execute the `<get-configuration-by-revision>` RPC, specify the revision identifier, and include the empty `<rollback-number/>` tag.

```
<rpc>
   <get-configuration-by-revision>
       <revision-string>re0-1605226203-331</revision-string>
       <rollback-number/>
   </get-configuration-by-revision>
</rpc>
```

The device returns the rollback index currently associated with that configuration revision identifier.

```
<rpc-reply>
<configuration-revision-information>
<rollback-number>2</rollback-number>
```

```
</configuration-revision-information>
</rpc-reply>
```

Similarly, to determine the configuration revision identifier currently associated with a specific rollback number, execute the <get-rollback-information> RPC, specify the rollback index, and include the empty <configuration-revision/> tag.

```
<rpc>
    <get-rollback-information>
        <rollback>2</rollback>
        <configuration-revision/>
    </get-rollback-information>
</rpc>
```

The device returns the configuration revision identifier currently associated with that rollback index.

```
<rpc-reply>
<rollback-information>
<configuration-revision>re0-1605226203-331</configuration-revision>
</rollback-information>
```

Retrieve a Configuration Using the Rollback Number

A Junos XML protocol client application can retrieve a previously committed (rollback) configuration using the rollback index. The client application executes the <get-rollback-information> RPC with the <rollback> element. The <rollback> element specifies the rollback index of the previous configuration to retrieve. The value can be from 0 (zero, for the most recently committed configuration) through one less than the number of stored previous configurations (maximum is 49). This operation is equivalent to the show system rollback operational mode command.

To request Junos XML-tagged output, which is the default, the application either includes the <format>xml</format> element or omits the <format> element.

```
<rpc>
    <get-rollback-information>
        <rollback>index-number</rollback>
    </get-rollback-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rollback-information>, and <configuration> elements. The <load-success/> tag is a side effect of the implementation and does not affect the results.

```
<rpc-reply xmlns:junos="URL">
    <rollback-information>
        <load-success/>
        <configuration attributes>
            <!-- tag elements for the complete previous configuration -->
         </configuration>
    </rollback-information>
</rpc-reply>
```

To request formatted ASCII output, the application includes the <format>text</format> element.

```
<rpc>
    <get-rollback-information>
        <rollback>index-number</rollback>
        <format>text</format>
    </get-rollback-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rollback-information>, <configuration-information>, and <configuration-output> elements.

```
<rpc-reply xmlns:junos="URL">
    <rollback-information>
        <load-success/>
        <configuration-information>
            <configuration-output>
                <!-- formatted ASCII text for the complete previous configuration -->
            </configuration-output>
        </configuration-information>
    </rollback-information>
</rpc-reply>
```

To request JSON format, the application includes the <format>json</format> element.

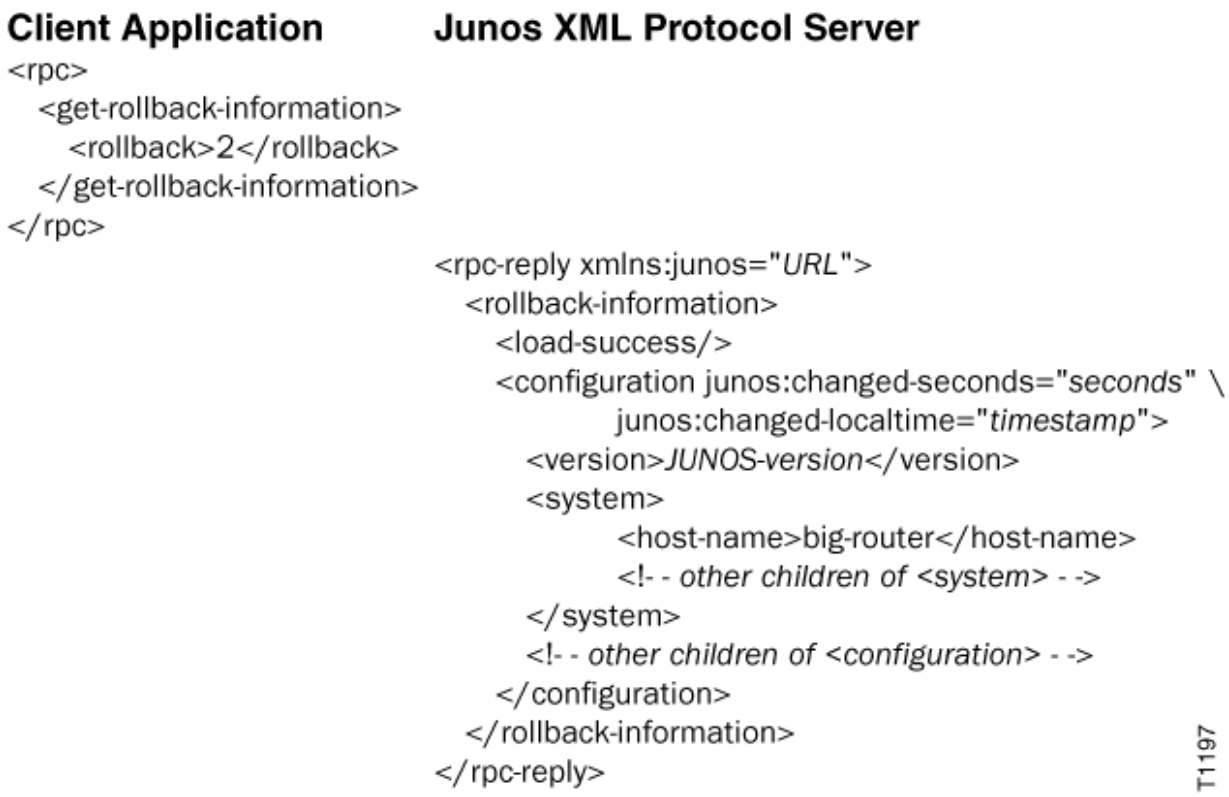
```
<rpc>
    <get-rollback-information>
```

```
        <rollback>index-number</rollback>
        <format>json</format>
    </get-rollback-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rollback-information>, <configuration-information>, and <json-output> elements.

```
<rpc-reply xmlns:junos="URL">
    <rollback-information>
        <load-success/>
        <configuration-information>
            <json-output>
                <!-- JSON data for the complete previous configuration -->
            </json-output>
        </configuration-information>
    </rollback-information>
</rpc-reply>
```

The following example requests Junos XML-tagged output for the rollback configuration that has an index of 2. In actual output, the *JUNOS-version* variable has a value such as 24.4R1, which is the initial version of Junos OS Release 24.4.



Retrieve a Configuration Using the Configuration Revision Identifier

When you successfully commit a configuration, Junos OS assigns that configuration a unique configuration revision identifier. Starting in Junos OS Release 20.4R1, a Junos XML protocol client application can use the `<get-configuration-by-revision>` RPC to retrieve the configuration by its configuration revision identifier. The `<revision-string>` value specifies the configuration revision ID. For example:

```
<rpc>
   <get-configuration-by-revision>
       <revision-string>re0-1605226203-331</revision-string>
   </get-configuration-by-revision>
</rpc>
```

The server returns the requested configuration enclosed in the `<configuration-revision-information>` element.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.4R1/junos">
<configuration-revision-information>
<load-success/>
<configuration junos:changed-seconds="1605226201" junos:changed-localtime="2020-11-12 16:10:01
```

```
PST">
...
</configuration>
</configuration-revision-information>
</rpc-reply>
```

By default, the <get-configuration-by-revision> RPC returns the configuration in XML format. To return the configuration as text or JSON format, include the <format> element and specify text or json.

```
<rpc>
    <get-configuration-by-revision>
        <revision-string>re0-1605226203-331</revision-string>
        <format>text</format>
    </get-configuration-by-revision>
</rpc>
```

```
<rpc>
    <get-configuration-by-revision>
        <revision-string>re0-1605226203-331</revision-string>
        <format>json</format>
    </get-configuration-by-revision>
</rpc>
```

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Compare the Active or Candidate Configuration to a Prior Version Using the Junos XML Protocol | 446
- Compare Two Previous (Rollback) Configurations Using the Junos XML Protocol | 450

Retrieve the Rescue Configuration Using the Junos XML Protocol

SUMMARY

A Junos XML protocol client application can use the <get-rescue-information> request tag to retrieve the existing rescue configuration on a Junos device.

A rescue configuration allows you to define a known working configuration or a configuration with a known state that you can restore at any time. You use the rescue configuration to revert to a known configuration or as a last resort if the device configuration and the backup configuration files become damaged beyond repair.

You must create a rescue configuration on the device before you can retrieve or use it. When you create a rescue configuration, the device saves the most recently committed configuration as the rescue configuration. You can create a rescue configuration using the following methods:

- In a Junos XML protocol session, use the <request-save-rescue-configuration> request tag.
- In the Junos OS CLI, issue the `request system configuration rescue save` operational mode command.

A Junos XML protocol client application can retrieve the rescue configuration from devices running Junos OS or devices running Junos OS Evolved. A client application requests the rescue configuration by emitting an <rpc> element and enclosing the Junos XML <get-rescue-information> request tag. The operation is equivalent to the `show system configuration rescue` operational mode command.

```
<rpc>
    <get-rescue-information/>
</rpc>
```

By default the server returns the rescue configuration in Junos XML format. To explicitly request XML output, the application can also include the <format>xml</format> element.

```
<rpc>
    <get-rescue-information>
        <format>xml</format>
    </get-rescue-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rescue-information>, and <configuration> tag elements. The <load-success/> tag is a side effect of the implementation and does not affect the results.

```
<rpc-reply xmlns:junos="URL">
    <rescue-information>
        <load-success/>
        <configuration attributes>
            <!-- tag elements representing the rescue configuration -->
        </configuration>
    </rescue-information>
</rpc-reply>
```

To request the rescue configuration in formatted ASCII output, the application includes the <format> element with the value text.

```
<rpc>
    <get-rescue-information>
        <format>text</format>
    </get-rescue-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rescue-information>, <configuration-information>, and <configuration-output> tag elements.

```
<rpc-reply xmlns:junos="URL">
    <rescue-information>
        <load-success/>
        <configuration-information>
            <configuration-output>
               <!-- formatted ASCII text representing the rescue configuration -->
            </configuration-output>
        </configuration-information>
    </rescue-information>
</rpc-reply>
```

To request the rescue configuration in JSON format, the application includes the <format> element with the value json.

```
<rpc>
    <get-rescue-information>
        <format>json</format>
    </get-rescue-information>
</rpc>
```

The Junos XML protocol server encloses its response in <rpc-reply>, <rescue-information>, <configuration-information>, and <json-output> elements.

```
<rpc-reply xmlns:junos="URL">
    <rescue-information>
        <load-success/>
        <configuration-information>
            <json-output>
                {
                  "configuration" : {
                     <!-- JSON data representing the rescue configuration -->
                  }
                }
            </json-output>
        </configuration-information>
    </rescue-information>
</rpc-reply>
```

RELATED DOCUMENTATION

Request Configuration Data Using the Junos XML Protocol | 375

Request the Complete Configuration Using the Junos XML Protocol | 414

Compare the Active or Candidate Configuration to a Prior Version Using the Junos XML Protocol

In the Junos OS CLI, you use the `compare` command to compare the active or candidate configuration to a previously committed configuration and display the differences. You can specify the comparison configuration by referencing its configuration revision identifier or its rollback number.

For example, in operational mode, you can compare the active configuration to a previously committed configuration by using the following commands:

- **show configuration | compare revision *revision-id***
- **show configuration | compare rollback *rollback-number***

In configuration mode, you can compare the candidate configuration to a previously committed configuration by using the following commands:

- **show | compare revision *revision-id***
- **show | compare rollback *rollback-number***

Similarly, in a Junos XML protocol session, a client application can compare the active or candidate configuration to a previously committed configuration (the comparison configuration). To display the differences, the client application uses the `<get-configuration>` operation with the `compare` attribute. The `compare` attribute accepts the following values, which indicate the method used to reference the comparison configuration:

- `configuration-revision`—Reference the comparison configuration by its configuration revision identifier string, which you define in the `configuration-revision="`*`revision-id`*`"` attribute.
- `rollback`—Reference the comparison configuration by its rollback index, which you define in the `rollback="`*`rollback-number`*`"` attribute.

You can combine the `compare` attribute with the `database` attribute to indicate whether to compare the candidate configuration or the active configuration to the previously committed configuration. The database attribute accepts the following values:

- `database="candidate"`—Compare the candidate configuration. This value is the default.
- `database="committed"`—Compare the active configuration.

You specify the comparison configuration by including either the `configuration-revision` or `rollback` attribute and specifying the appropriate configuration revision identifier or rollback index. If you do not provide the `configuration-revision` or `rollback` attribute or if you provide an invalid configuration revision identifier, the server uses the active configuration as the comparison configuration. The active configuration corresponds to rollback number 0.

For example, to compare the candidate configuration to the configuration that has the given configuration revision identifier, use the following syntax:

```
<rpc>
    <get-configuration compare="configuration-revision" configuration-revision="revision-id"
format="text">
        <!-- optional - configuration elements to compare -->
    </get-configuration>
</rpc>
```

Similarly, to compare the candidate configuration to the configuration that has the given rollback index, use the following syntax:

```
<rpc>
    <get-configuration compare="rollback" rollback="[0-49]" format="text">
        <!-- optional - configuration elements to compare -->
    </get-configuration>
</rpc>
```

To compare the active configuration to a previous configuration, include the `database="committed"` attribute. For example:

```
<rpc>
    <get-configuration database="committed" compare="configuration-revision" configuration-
revision="re0-1605138555-328"/>
</rpc>
```

You can also specify the scope of the comparison. You can compare the full configuration, or you can compare a subset of the configuration. To compare a subset of the configuration, define a subtree filter that selects the elements to compare, as shown in the following example:

```
<rpc>
  <get-configuration compare="configuration-revision" configuration-
revision="re0-1605288042-335">
    <configuration>
      <system>
        <scripts/>
      </system>
    </configuration>
```

```
  </get-configuration>
</rpc>
```

When you compare the candidate configuration to the active configuration, the `compare` operation returns XML output. For all other comparisons, it returns the output as text using a patch format. The `<configuration-information>` and `<configuration-output>` tags enclose the text output. The output uses the following conventions to specify the differences between configurations:

- Statements that are only in the active or candidate configuration are prefixed with a plus sign (+).
- Statements that are only in the comparison file are prefixed with a minus sign (–).
- Statements that are unchanged are prefixed with a single blank space ( ).

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.4R1/junos">
<configuration-information>
<configuration-output>
[edit system scripts op]
-     file bgp-summary.py;
+     file bgp-neighbors.py;
</configuration-output>
</configuration-information>
</rpc-reply>
```

When you compare the candidate configuration to the active configuration, you can display the differences in text, XML, or JSON format. To select a format, include the `format` attribute in the request and specify the format value.

For example, the following request returns XML format:

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

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R1/junos">
<configuration>
    <system>
        <scripts operation="create">
            <op>
                <file>
                    <name>bgp.slax</name>
                </file>
            </op>
```

```
        </scripts>
    </system>
</configuration>
</rpc-reply>
```

The following request returns JSON format:

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

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R1/junos">
<configuration-information>
<json-output>
{
    "configuration" : {
        "system" : {
            "scripts" : {
                "@" : {
                    "operation" : "create"
                },
                "op" : {
                    "file" : [
                    {
                        "name" : "bgp.slax"
                    }
                    ]
                }
           }
         }
    }
}
</json-output>
</configuration-information>
</rpc-reply>
```

NOTE: When you compare the candidate and active configurations and display the differences in XML or JSON format, the device omits the `<configuration>` tag in the XML output and omits the `configuration` object in the JSON output in the following cases:

- The comparison returns no differences
- The comparison returns differences for only non-native configuration data, for example, configuration data associated with an OpenConfig data model.

RELATED DOCUMENTATION

Compare Two Previous (Rollback) Configurations Using the Junos XML Protocol | 450

Compare Two Previous (Rollback) Configurations Using the Junos XML Protocol

In a Junos XML protocol session with a device running Junos OS, a client application can compare the contents of two previously committed (rollback) configurations by using either of the following RPCs with the <compare> element:

- <get-configuration-by-revision>—Compare configurations by referencing a configuration revision identifier.
- <get-rollback-information>—Compare configurations by referencing a rollback index.

The <get-configuration-by-revision> RPC with the <compare> element is equivalent to the show system configuration revision operational mode command with the compare option. The <compare> element specifies the configuration revision identifier of the configuration that is the basis for comparison. The <revision-string> element defines the configuration revision identifier of the configuration to compare with the base configuration. The syntax is:

```
<rpc>
    <get-configuration-by-revision>
        <revision-string>revision-id</revision-string>
        <compare>revision-id</compare>
    </get-configuration-by-revision>
</rpc>
```

For example, the following RPC compares two configurations by referencing their configuration revision identifier strings:

```
<rpc>
    <get-configuration-by-revision>
        <revision-string>re0-1605288042-335</revision-string>
        <compare>re0-1605288033-334</compare>
    </get-configuration-by-revision>
</rpc>
```

Similarly, the <get-rollback-information> RPC with the <compare> element is equivalent to the show system rollback operational mode command with the compare option. The <compare> element specifies the rollback index of the configuration that is the basis for comparison. The <rollback> element specifies the rollback index 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. The syntax is:

```
<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 is the base configuration. Its index is enclosed in the <compare> element, and the index of the more recent configuration is enclosed in the <rollback> or <revision-string> tag element.

The Junos XML protocol server encloses its response in an <rpc-reply> element, a <rollback-information> or <configuration-revision-information> element, depending on the RPC, and <configuration-information> and <configuration-output> elements. The <load-success/> tag is a side effect of the implementation and does not affect the results.

```
<rpc-reply xmlns:junos="URL">
    <rollback-information>
        <load-success/>
        <configuration-information>
            <configuration-output>
                <!-- formatted ASCII text representing the changes -->
```

```
            </configuration-output>
        </configuration-information>
    </rollback-information>
</rpc-reply>
```

The information in the <configuration-output> tag element is formatted ASCII text 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.

The following example shows how to request a comparison of the rollback configurations that have indexes of 20 and 4.

Client Application

```
<rpc>
  <get-rollback-information>
    <rollback>20</rollback>
    <compare>4</compare>
  </get-rollback-information>
</rpc>
```

Junos XML Protocol Server

```
<rpc-reply xmlns:junos="URL">
  <rollback-information>
    <load-success/>
    <configuration-information>
      <configuration-output>
        [edit interfaces]
        -     ge-0/2/0 {
        -         stacked-vlan-tagging;
        -         mac 00.01.02.03.04.05;
        -         gigether-options {
        -             loopback;
        -         }
        -     }
        [edit]
        +     services {
        +         l2tp {
        +             tunnel-group 12 {
        +                 local-gateway;
        +             }
        +         }
        +     }
      </configuration-output>
    </configuration-information>
  </rollback-information>
</rpc-reply>
```

T1170

RELATED DOCUMENTATION

- Request Configuration Data Using the Junos XML Protocol | 375
- Specify the Scope of the Configuration Data to Return | 387
- Request the Complete Configuration Using the Junos XML Protocol | 414
- Retrieve a Previous (Rollback) Configuration Using the Junos XML Protocol | 435
- Retrieve the Rescue Configuration Using the Junos XML Protocol | 443

Compare the Active or Candidate Configuration to a Prior Version Using the Junos XML Protocol | 446

Request an XML Schema for the Configuration Hierarchy Using the Junos XML Protocol

IN THIS SECTION

The Junos configuration schema represents all configuration elements available in the version of the OS that is running on a device. To determine the Junos OS or Junos OS Evolved version, emit the <get-software-information> operational request tag. Client applications can use the schema simply to learn which configuration statements are available in their version of Junos OS or Junos OS Evolved. Client applications can also use the schema to validate the configuration on a device.

The schema does not indicate which elements are actually configured. Moreover, the schema does not indicate that you can even configure an element 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-configuration> tag element instead.

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*. The primer provides a basic introduction and lists the formal specifications where you can find detailed information.

Request an XML Schema for the Configuration Hierarchy

A Junos XML protocol client application can request an XML Schema-language representation of the entire configuration hierarchy on a device running Junos OS or a device running Junos OS Evolved. To request the XML schema, a client application emits an <rpc> element and encloses the Junos XML <get-xnm-information> element. The <get-xnm-information> element encloses the <type> and <namespace> child elements with the indicated values.

```
<rpc>
    <get-xnm-information>
```

```
        <type>xml-schema</type>
        <namespace>junos-configuration</namespace>
    </get-xnm-information>
</rpc>
```

The Junos XML protocol server encloses the XML schema in <rpc-reply> and <xsd:schema> tags.

```
<rpc-reply xmlns:junos="URL">
    <xsd:schema>
        <!-- tag elements for the Junos XML schema -->
    </xsd:schema>
</rpc-reply>
```

Create 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.

The <xsd:schema> element encloses the <xsd:import> tag, which references the **junos.xsd** file. This file contains the required information about the junos namespace. For example, the following <xsd:import> tag specifies the file for Junos OS Release 20.4R1:

```
<xsd:import schemaLocation="junos.xsd" namespace="http://xml.juniper.net/junos/20.4R1/junos"/>
```

To enable the schema validator to interpret the <xsd:import> tag, the **junos.xsd** file must exist. You must manually create a file called **junos.xsd** in the directory where the Junos configuration schema resides. 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 or Junos OS Evolved release running on the device (for example, 20.4R1).

```
<?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: Request an XML Schema

The following examples request the Junos OS configuration schema. In the Junos XML protocol server's response, the first <xsd:element> statement defines the <undocumented> Junos XML tag element. This element 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 server's response appear on multiple lines for legibility only. Also, in actual output the *JUNOS-version* variable is replaced by a value such as 20.4R1 for the initial version of Junos OS Release 20.4.

Client Application **Junos XML Protocol Server**

```
<rpc>
  <get-xnm-information>
    <type>xml-schema</type>
    <namespace>junos-configuration</namespace>
  </get-xnm-information>
</rpc>
```

```
<rpc-reply 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>
```

T1177

Another <xsd:element> statement near the beginning of the schema defines the Junos XML <configuration> element. It encloses the <xsd:element> statement that defines the <system> element, which corresponds to

the `[edit system]` hierarchy level. For brevity, the output omits the statements corresponding to other hierarchy levels.

Client Application **Junos XML Protocol 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> - ->
                </xsd:choice >
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <!- - definitions for other hierarchy levels  - ->
        </xsd:choice >
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
</rpc-reply>
```

T1178

RELATED DOCUMENTATION

Understanding the Request Procedure in a Junos XML Protocol Session | 57

Request Configuration Data Using the Junos XML Protocol | 375

PART 5

Junos XML Protocol Utilities

CHAPTER 16

Develop Junos XML Protocol C Client Applications

IN THIS CHAPTER

Establish a Junos XML Protocol Session Using C Client Applications

This example illustrates how a Junos XML protocol C client application uses the SSH or Telnet protocol to establish a connection and Junos XML protocol session with a device running Junos OS. In the line that begins with the string `execlp`, the client application invokes the `ssh` command. (Substitute the `telnet` command if appropriate.) The *routing-platform* argument to the `execlp` routine specifies the hostname or IP address of the Junos XML protocol server device. The `junoscript` argument is the command that converts the connection to a Junos XML protocol session.

```
int ipipes[ 2 ], opipes[ 2 ];
pid_t pid;
int rc;
char buf[ BUFSIZ ];

if (pipe(ipipes) <0 || pipe(opipes) <0)
    err(1, "pipe failed");

pid = fork( );
if (pid <0)
    err(1, "fork failed");

if (pid == 0) {
    dup2(opipes[ 0 ], STDIN_FILENO);
    dup2(ipipes[ 1 ], STDOUT_FILENO);
    dup2(ipipes[ 1 ], STDERR_FILENO);
    close(ipipes[ 0 ]); /* close read end of pipe */
```

```
        close(ipipes[ 1 ]); /* close write end of pipe */
        close(opipes[ 0 ]); /* close read end of pipe */
        close(opipes[ 1 ]); /* close write end of pipe */

        execlp("ssh", "ssh", "-x",  routing-platform  , "junoscript", NULL);
        err (1, "unable to execute: ssh %s junoscript," device);
    }

    close(ipipes[ 1 ]); /* close write end of pipe */
    close(opipes[ 0 ]); /* close read end of pipe */
```

```
    if (write(opipes[ 1 ], initial_handshake, strlen(initial_handshake)) <0 )
        err(1, "writing initial handshake failed");

    rc=read(ipipes[ 0 ], buf, sizeof(buf));
    if (rc <0)
         err(1, "read initial handshake failed");
```

RELATED DOCUMENTATION

Access and Edit Device Configurations Using Junos XML Protocol C Client Applications | 460

Access and Edit Device Configurations Using Junos XML Protocol C Client Applications

This example script presents a C client application that can be used to access, edit, and commit configurations on routers, switches, and security devices running Junos OS

```
//--Includes--//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
#include <fcntl.h>
#include <errno.h>
#include <libxml/parser.h>
#include <libxml/xpath.h>

//--Defines--//
//#define PRINT
     //--Toggles printing of all data to and from js server--//

//--Global Variables and Initialization--//
int sockfd;
char *xmlns_start_ptr = NULL;
char *xmlns_end_ptr = NULL;
int sock_bytes, pim_output_len, igmp_output_len, count_a, count_x, count_y,
          count_z, repl_str_len, orig_len, up_to_len, remain_len, conf_chg;
struct sockaddr_in serv_addr;
struct hostent *server;
char temp_buff[1024];           //--Temporary buffer used when --//
                                //--sending js configuration commands--//
char rcvbuffer[255];            //--Stores data from socket--//
char *pim_output_ptr = NULL;    //--Pointer for pim_output from socket--//
                                //--buffer--//
char *igmp_output_ptr = NULL;   //--Pointer for igmp_output from socket buffer--//
char small_buff[2048];          //--Buffer to support js communication--//
char jserver[16];               //--Junos XML protocol server IP address--//
int jport = 3221;               //--Junos XML protocol server port --//
                                //--(xnm-clear-text)--//
char msource[16];               //--Multicast source of group being
                                //--configured under igmp--//
char minterface[16];            //--Local multicast source interface--//
                                //--###change in igmp_xpath_ptr as well###--//
xmlDocPtr doc;                  //--Pointer struct for parsing XML--//
xmlChar *pim_xpath_ptr =
          (xmlChar*) "/rpc-reply/pim-join-information/join-family
           /join-group[upstream-state-flags/local-source]
           /multicast-group-address";
xmlChar *temp_xpath_ptr =
          (xmlChar*) "/rpc-reply/igmp-group-information
          /mgm-interface-groups/mgm-group
          [../interface-name = '%s']/multicast-group-address";
xmlChar *igmp_xpath_ptr = NULL;
xmlNodeSetPtr nodeset;
xmlXPathObjectPtr pim_result;   //--Pointer for pim result xml parsing--//
```

```
xmlXPathObjectPtr igmp_result;  //--Pointer for igmp result xml parsing--//
xmlChar *keyword_ptr = NULL;    //--Pointer for node text--//
char pim_result_buff[128][64];  //--Char array to store pim xPath results--//
char igmp_result_buff[128][64]; //--Char array to store igmp xPath results--//

//--js commands-//
char js_handshake1[64] = "<?xml version=\"1.0\" encoding=\"us-ascii\"?>\n";
char js_handshake2[128] = "<junoscript version=\"1.0\"
          hostname=\"client1\" release=\"8.4R1\">\n";
char js_login[512] = "<rpc>\n<request-login>\n<username>lab</username>
          \n<challenge-response>Lablab</challenge-response>
          \n</request-login>\n</rpc>\n";
char js_show_pim[512] = "<rpc>\n<get-pim-join-information>
          \n<extensive/></get-pim-join-information></rpc>\n";
char js_show_igmp[512] = "<rpc>\n<get-igmp-group-information/>\n</rpc>\n";
char js_rmv_group[512] = "<rpc>\n<load-configuration>\n<configuration>
          \n<protocols>\n<igmp>\n<interface>\n<name>%s</name>
          \n<static>\n<group delete='delete'>\n<name>%s</name>
          \n</group>\n</static>\n</interface>\n</igmp>\n</protocols>
          \n</configuration>\n</load-configuration>\n</rpc>\n\n\n\n\n";
char js_add_group[512] = "<rpc>\n<load-configuration>
          \n<configuration>\n<protocols>\n<igmp>
          \n<interface>\n<name>%s</name>\n<static>
          \n<group>\n<name>%s</name>\n<source>
          \n<name>%s</name>\n</source>\n</group>\n</static>
          \n</interface>\n</igmp>\n</protocols>\n</configuration>
          \n</load-configuration>\n</rpc>\n";
char js_commit[64] = "<rpc>\n<commit-configuration/>\n</rpc>\n";

//--Function prototypes--//
void error(char *msg);          //--Support error messaging--//
xmlDocPtr getdoc(char *buffer); //--Parses XML content and loads it into memory--//
xmlXPathObjectPtr getnodeset (xmlDocPtr doc, xmlChar *xpath);
          //--Parses xml content for result node(s) from XPath search--//

//--Functions--//
void error(char *msg) {
  perror(msg);
  exit(0);
}

xmlDocPtr getdoc(char *buffer) {
```

```
  xmlDocPtr doc;

  doc = xmlReadMemory(buffer, strlen((char *)buffer), "temp.xml", NULL, 0);
    if (doc == NULL ) {
      fprintf(stderr,"Document not parsed successfully. \n");
      return NULL;
    } else {
      #ifdef PRINT
      printf("Document parsed successfully. \n");
      #endif
    }
    return doc;
  }

  xmlXPathObjectPtr getnodeset (xmlDocPtr doc, xmlChar *xpath) {

    xmlXPathContextPtr context;
    xmlXPathObjectPtr result;

    context = xmlXPathNewContext(doc);
    if (context == NULL) {
      printf("Error in xmlXPathNewContext\n");
      return NULL;
    }
    result = xmlXPathEvalExpression(xpath, context);
    xmlXPathFreeContext(context);
    if (result == NULL) {
      printf("Error in xmlXPathEvalExpression\n");
      return NULL;
    }
    if(xmlXPathNodeSetIsEmpty(result->nodesetval)) {
      xmlXPathFreeObject(result);
      #ifdef PRINT
      printf("No result\n");
      #endif
      return NULL;
    }
    return result;
  }

//--Main--//
  int main(int argc, char **argv) {
```

```
  if(argc != 4) {
    printf("\nUsage: %s <device Address> <Interface Name>
            <Multicast Source>\n\n", argv[0]);
    exit(0);
  } else {
    strcpy(jserver, argv[1]);
    strcpy(minterface, argv[2]);
    strcpy(msource, argv[3]);
  }
  igmp_xpath_ptr = (xmlChar *)realloc((xmlChar *)igmp_xpath_ptr, 1024);
  sprintf(igmp_xpath_ptr, temp_xpath_ptr, minterface);

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  server = gethostbyname(jserver);
  bzero((char*) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char*) server->h_addr, (char*)
            &serv_addr.sin_addr.sin_addr, server->h_length);
  serv_addr.sin_port = htons(jport);

  //--Connect to the js server--//
  if(connect(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
    printf("Socket connect error\n");
  }

  if(fcntl(sockfd, F_SETOWN, getpid()) < 0)
  error("Unable to set process owner to us\n");
  printf("\nConnected to %s on port %d\n", jserver, jport);

//--Read data from the initial connect--//
  sock_bytes = read(sockfd, rcvbuffer, 255);
  #ifdef PRINT
  printf("\n%s", rcvbuffer);
  #endif

//--js intialization handshake--//
  sock_bytes = write(sockfd, js_handshake1, strlen(js_handshake1));
            //--Send xml PI to js server--//
  sock_bytes = write(sockfd, js_handshake2, strlen(js_handshake2));
            //--Send xml version and encoding to js server--//
  sock_bytes = read(sockfd, rcvbuffer, 255);
            //--Read return data from sock buffer--//
  rcvbuffer[sock_bytes] = 0;
```

```
  printf("XML connection to the Junos XML protocol server has been initialized\n");
  #ifdef PRINT
  printf("\n%s", rcvbuffer);
  #endif

//--js login--//
  sock_bytes = write(sockfd, js_login, strlen(js_login));
            //--Send js command--//
  while(strstr(small_buff, "superuser") == NULL) {
            //--Continue to read from the buffer until match--//
    sock_bytes = read(sockfd, rcvbuffer, 255);
    rcvbuffer[sock_bytes] = 0;
    strcat(small_buff, rcvbuffer);
              //--Copy buffer contents into pim_buffer--//
  }
  printf("Login completed to the Junos XML protocol server\n");
  #ifdef PRINT
  printf("%s\n", small_buff);  //--Print the small buff contents--//
  #endif
//regfree(&regex_struct);
  bzero(small_buff, strlen(small_buff));
            //--Erase small buffer contents--//

//--Begin the for loop here--//
  printf("Running continuous IGMP and PIM group comparison...\n\n");
  for(;;) {                   //--Begin infinite for loop--//

//--Get PIM join information--//
    pim_output_ptr = (char *)realloc((char *)pim_output_ptr,
              strlen(js_handshake1));
              //--Allocate memory for xml PI concatenation --//
              //--to pim_output_ptr--//
    strcpy(pim_output_ptr, js_handshake1);
              //--Copy PI to pim_output_ptr--//
    sock_bytes = write(sockfd, js_show_pim, strlen(js_show_pim));
              //--Send show pim joins command--//
    while(strstr(pim_output_ptr, "</rpc-reply>") == NULL) {
              //--Continue to read from the buffer until match--//
      sock_bytes = read(sockfd, rcvbuffer, 255);
                //--Read from buffer--//
      rcvbuffer[sock_bytes] = 0;
      pim_output_len = strlen((char *)pim_output_ptr);
                //--Determine current string length of pim_output_ptr--//
```

```
      pim_output_ptr = (char *)realloc((char *)pim_output_ptr,
                strlen(rcvbuffer)+pim_output_len);
                //--Reallocate memory for additional data--//
      strcat(pim_output_ptr, rcvbuffer);
                //--Copy data from rcvbuffer to pim_output_ptr--//
    }

//--Remove the xmlns entry--//
    xmlns_start_ptr = strstr(pim_output_ptr, "xmlns=\"http:");
              //--Find the start of the xmlns entry--pointer --//
              //--returned by strstr()--//
    xmlns_end_ptr = strstr(xmlns_start_ptr, ">");
              //--Find the end of the xmlns entry--pointer --//
              //--returned by strstr()--//
    repl_str_len = xmlns_end_ptr - xmlns_start_ptr;
              //--Determine the length of the string to be replaced--//
    orig_len = strlen((char *)pim_output_ptr) + 1;
              //--Determine the original length of pim_output--//
    up_to_len = xmlns_start_ptr - pim_output_ptr;
              //--Determine the length up to the beginning --//
              //--of the xmlns entry--//
    remain_len = orig_len - (up_to_len + repl_str_len);
              //--Determine what the remaining length is minus --//
              //--what we are removing--//
    memcpy(xmlns_start_ptr - 1, xmlns_start_ptr + repl_str_len, remain_len);
              //--copy the remaining string to the beginning --//
              //--of the replacement string--//
    #ifdef PRINT
    printf("\n%s\n", pim_output_ptr);
    #endif
//--End of GET PIM join information--//

//--Get IGMP membership information--//
    igmp_output_ptr = (char *)realloc((char *)igmp_output_ptr,
              strlen(js_handshake1));
    strcpy(igmp_output_ptr, js_handshake1);
    sock_bytes = write(sockfd, js_show_igmp, strlen(js_show_igmp));
    while(strstr(igmp_output_ptr, "</rpc-reply>") == NULL) {
      sock_bytes = read(sockfd, rcvbuffer, 255);
      rcvbuffer[sock_bytes] = 0;
      igmp_output_len = strlen((char *)igmp_output_ptr);
      igmp_output_ptr = (char *)realloc((char *)igmp_output_ptr,
                strlen(rcvbuffer)+igmp_output_len);
```

```
      strcat(igmp_output_ptr, rcvbuffer);
    }
    #ifdef PRINT
    printf("\n%s\n", igmp_output_ptr);
    #endif
//--End of GET IGMP membership information--//

//--Store xPath results for pim buffer search--//
    doc = getdoc(pim_output_ptr);
              //--Call getdoc() to parse XML in pim_output--//
    pim_result = getnodeset (doc, pim_xpath_ptr);
              //--Call getnodeset() which provides xPath result--//
    if (pim_result) {
      nodeset = pim_result->nodesetval;
      for (count_a=0; count_a < nodeset->nodeNr; count_a++) {
                //--Run through all node values found--//
        keyword_ptr = xmlNodeListGetString
                  (doc, nodeset->nodeTab[count_a]->xmlChildrenNode, 1);
        strcpy(pim_result_buff[count_a], (char *)keyword_ptr);
                  //--Copy each node value to its own array element--//
      #ifdef PRINT
      printf("PIM Groups: %s\n", pim_result_buff[count_a]);
                //--Print the node value--//
      #endif

      xmlFree(keyword_ptr);     //--Free memory used by keyword_ptr--//
      xmlChar *keyword_ptr = NULL;
      }
    xmlXPathFreeObject(pim_result);
              //--Free memory used by result--//
  }
  xmlFreeDoc(doc);              //--Free memory used by doc--//
  xmlCleanupParser();           //--Clean everything else--//
//--End of xPath search--//

//--Store xPath results for igmp buffer search--//
  doc = getdoc(igmp_output_ptr);
  igmp_result = getnodeset (doc, igmp_xpath_ptr);
  if (igmp_result) {
    nodeset = igmp_result->nodesetval;
    for (count_a=0; count_a < nodeset->nodeNr; count_a++) {
      keyword_ptr = xmlNodeListGetString
                (doc, nodeset->nodeTab[count_a]->xmlChildrenNode, 1);
```

```
      strcpy(igmp_result_buff[count_a], (char *)keyword_ptr);
      #ifdef PRINT
      printf("IGMP Groups: %s\n", igmp_result_buff[count_a]);
      #endif

      xmlFree(keyword_ptr);
      xmlChar *keyword_ptr = NULL;
    }
    xmlXPathFreeObject(igmp_result);
  }
  xmlFreeDoc(doc);
  xmlCleanupParser();
//--End of xPath search--//

//--Code to compare pim groups to configured igmp static membership--//
  conf_chg = 0;
  count_x=0;                  //--Track pim groups--//
  count_y=0;                  //--Track igmp groups--//
  count_z=0;          //--Track matches (if set to 1, igmp group matched pim group)--//
  while(strstr(pim_result_buff[count_x], "2") != NULL) {
           //--Run through igmp pim groups--//
    if(strstr(igmp_result_buff[count_y], "2") == NULL) {
      count_z = 0;
      conf_chg = 1;
    }
    while(strstr(igmp_result_buff[count_y], "2") != NULL) {
            //--For each pim group, run through all igmp groups--//
      if(strcmp(igmp_result_buff[count_y], pim_result_buff[count_x]) == 0) {
                //--If igmp group matches pim group, set z to 1 --//
                //-- (ie  count_z=1; --//
                //--Set z to 1 if there was a match (ie - the static --//
                //--membership is configured)--//
      }
      count_y++;              //--Increment igmp result buffer--//
    }
    if(count_z == 0) {        //--If no igmp group matched the --//
                              //--pim group (z stayed at 0), configure--//
                              //--static membership--//
      printf("Adding this group to igmp: %s\n", pim_result_buff[count_x]);
      sprintf(temp_buff, js_add_group, minterface,
              pim_result_buff[count_x], msource);
              //--Copy js_add_group with pim group to temp_buff--//
      #ifdef PRINT
```

```
        printf("%s", temp_buff);
        #endif
        sock_bytes = write(sockfd, temp_buff, strlen(temp_buff));
        while(strstr(small_buff, "</rpc-reply>") == NULL) {
          sock_bytes = read(sockfd, rcvbuffer, 255);
          rcvbuffer[sock_bytes] = 0;
          strcat(small_buff, rcvbuffer);
        }
        #ifdef PRINT
        printf("%s\n", small_buff);
        #endif
        bzero(small_buff, strlen(small_buff));
                  //--Erase (copy all 0's) small buffer contents--//
        bzero(temp_buff, strlen(temp_buff));
                  //--Erase temp_buff contents--//
        conf_chg = 1;
                  //--Set conf_chg value to 1 to signify that a --//
                  //--commit is needed--//
      }
      count_x++;                //--increment pim result buffer--//
      count_y=0;                //--reset igmp result buffer to start--//
                                //-- at first element--//
      count_z=0;                //--reset group match to 0 --//
                                //--(config needed due to no match)--/
    }

  //--Code for comparing igmp static membership to pim groups--//
    count_x=0;
    count_y=0;
    count_z=0;
    while(strstr(igmp_result_buff[count_y], "2") != NULL) {
      if(strstr(pim_result_buff[count_x], "2") == NULL) {
        count_z = 0;
        conf_chg = 1;
      }
      while(strstr(pim_result_buff[count_x], "2") != NULL) {
        if(strcmp(pim_result_buff[count_x], igmp_result_buff[count_y]) == 0) {
          count_z = 1;
        }
        count_x++;
      }
      if(count_z == 0) {
        printf("Removing this group from igmp: %s\n", igmp_result_buff[count_y]);
```

```
        sprintf(temp_buff, js_rmv_group, minterface, igmp_result_buff[count_y]);
        #ifdef PRINT
        printf("%s", temp_buff);
        #endif
        sock_bytes = write(sockfd, temp_buff,
strlen(temp_buff));
        while(strstr(small_buff, "</rpc-reply>") == NULL) {
          sock_bytes = read(sockfd, rcvbuffer, 255);
          rcvbuffer[sock_bytes] = 0;
          strcat(small_buff, rcvbuffer);
        }
        #ifdef PRINT
        printf("%s\n", rcvbuffer);
        #endif
        bzero(small_buff, strlen(small_buff));
        bzero(temp_buff, strlen(temp_buff));
        conf_chg = 1;
      }
      count_y++;
      count_x=0;
      count_z=0;
    }

    if(conf_chg == 1) {
      sock_bytes = write(sockfd, js_commit, strlen(js_commit));
      while(strstr(small_buff, "</rpc-reply>") == NULL) {
        sock_bytes = read(sockfd, rcvbuffer, 255);
        rcvbuffer[sock_bytes] = 0;
        strcat(small_buff, rcvbuffer);
      }
      bzero(small_buff, strlen(small_buff));
      printf("\nCommitted configuration change\n");
    } else {
      #ifdef PRINT
      printf("\nNo configuration changes made\n");
      #endif
    }
    #ifdef PRINT
    printf("\n%s\n", small_buff);
    #endif

  //--Cleanup before next round of checks--//
    bzero(rcvbuffer, strlen(rcvbuffer));
```

```
            //--Erase contents of rcvbuffer--//
    char *xmlns_start_ptr = NULL;
            //--Nullify the contents--//
    char *xmlns_end_ptr = NULL;
            //--Nullify the contents--//
    for(count_x = 0; count_x < 129; count_x++) {
            //--Erase contents of both pim_result_buff and igmp_result_buff--//
      bzero(pim_result_buff[count_x], strlen(pim_result_buff[count_x]));
      bzero(igmp_result_buff[count_x], strlen(igmp_result_buff[count_x]));
    }
  }
}
```

RELATED DOCUMENTATION

Establish a Junos XML Protocol Session Using C Client Applications | 459

PART 6

Configuration Statements and Operational Commands

Junos CLI Reference Overview

We've consolidated all Junos CLI commands and configuration statements in one place. Read this guide to learn about the syntax and options that make up the statements and commands. Also understand the contexts in which you'll use these CLI elements in your network configurations and operations.

- Junos CLI Reference

Click the links to access Junos OS and Junos OS Evolved configuration statement and command summary topics.

- Configuration Statements
- Operational Commands