SRC PE Software

NETCONF API Guide

Release
4.12.x

Modified: 2018-10-15

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

SRC PE Software NETCONF API Guide
Release 4.12.x

Revision History
October 2018—Revision 1

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

YEAR 2000 NOTICE

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

SOFTWARE LICENSE

The terms and conditions for using this software are described in the software license contained in the acknowledgment to your purchase order or, to the extent applicable, to any reseller agreement or end-user purchase agreement executed between you and Juniper Networks. By using this software, you indicate that you understand and agree to be bound by those terms and conditions.

Generally speaking, the software license restricts the manner in which you are permitted to use the software and may contain prohibitions against certain uses. The software license may state conditions under which the license is automatically terminated. You should consult the license for further details.

For complete product documentation, please see the Juniper Networks Web site at www.juniper.net/techpubs.

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.

Abbreviated Table of Contents

Table of Contents

List of Tables

About the Documentation

SRC Documentation and Release Notes

For a list of related SRC documentation, see https://www.juniper.net/documentation/.

If the information in the latest *SRC Release Notes* differs from the information in the SRC guides, follow the *SRC Release Notes*.

Audience

This documentation is intended for experienced system and network specialists working with routers running Junos OS and JunosE software in an Internet access environment. We assume that readers know how to use the routers, directories, and RADIUS servers that they will deploy in their SRC networks. If you are using the SRC software in a cable network environment, we assume that you are familiar with the PacketCable Multimedia Specification (PCMM) as defined by Cable Television Laboratories, Inc. (CableLabs) and with the Data-over-Cable Service Interface Specifications (DOCSIS) 1.1 protocol. We also assume that you are familiar with operating a multiple service operator (MSO) multimedia-managed IP network.

Documentation Conventions

Table 1 on page xii defines the notice icons used in this guide. Table 2 on page xii defines text conventions used throughout this documentation.

Table 1: Notice Icons

Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2: Text Conventions

Convention	Description	Examples
Bold text like this	• Represents keywords, scripts, and tools in text. • Represents a GUI element that the user selects, clicks, checks, or clears.	• Specify the keyword **exp-msg**. • Run the **install.sh** script. • Use the **pkgadd** tool. • To cancel the configuration, click **Cancel**.
Bold text like this	Represents text that the user must type.	**user@host# set cache-entry-age *cache-entry-age***

`Fixed-width text like this`	Represents information as displayed on your terminal's screen, such as CLI commands in output displays.	`nic-locators {`
		`  login {`
		`  resolution {`
		`  resolver-name /realms/`
		`  login/A1;`
		`  key-type LoginName;`
		`  value-type SaeId;`
		`  }`

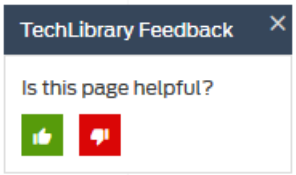
Table 2: Text Conventions (continued)

Regular sans serif typeface	• Represents configuration statements. • Indicates SRC CLI commands and options in text. • Represents examples in procedures. • Represents URLs.	• **system ldap server{ stand-alone;** • Use the **request sae modify device failover** command with the **force** option • **user@host# . . .** • **https://www.juniper.net/documentation/software/management/src/api-index.html**
Italic sans serif typeface	Represents variables in SRC CLI commands.	**user@host# set local-address *local-address***
Angle brackets	In text descriptions, indicate optional keywords or variables.	Another runtime variable is <gfwif>.
Key name	Indicates the name of a key on the keyboard.	Press Enter.
Key names linked with a plus sign (+)	Indicates that you must press two or more keys simultaneously.	Press Ctrl + b.
Italic typeface	• Emphasizes words. • Identifies book names. • Identifies distinguished names. • Identifies files, directories, and paths in text but not in command examples.	• There are two levels of access: *user* and *privileged*. • *SRC PE Getting Started Guide* • *o=Users, o=UMC* • The */etc/default.properties* file.
Backslash	At the end of a line, indicates that the text wraps to the next line.	Plugin.radiusAcct-1.class=\ net.juniper.smgt.sae.plugin\ RadiusTrackingPluginEvent
Words separated by the | symbol	Represent a choice to select one keyword or variable to the left or right of this symbol. (The keyword or variable may be either optional or required.)	diagnostic | line

Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can provide feedback by using either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the Juniper Networks TechLibrary site, and do one of the following:



  - Click the thumbs-up icon if the information on the page was helpful to you.

- Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.

- E-mail—Send your comments to **techpubs-comments@juniper.net**. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or Partner Support Service support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

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

Self-Help Online Tools and Resources

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

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

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

Opening a Case with JTAC

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

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

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

PART 1

Using the SRC XML and NETCONF APIs

CHAPTER 1

Introduction to the SRC XML and NETCONF APIs

The *NETCONF API* (application programming interface) is an Extensible Markup Language (XML) application that client applications use to request and change configuration information on a C Series Controller that runs the SRC software. The operations defined in the API are equivalent to configuration mode commands in the SRC command-line interface (CLI). Applications use the API to display, edit, and commit configuration statements (among other operations), just as administrators use CLI configuration mode commands such as **show**, **set**, and **commit** to perform those operations.

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

The NETCONF API is described in RFC 4741, *NETCONF Configuration Protocol*, available at http://www.ietf.org/rfc/rfc4741.txt.

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

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

This chapter includes the following topics:

About XML

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

The following sections discuss XML and NETCONF:

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

XML and NETCONF Tag Elements

Items in an XML-compliant document or data set are always enclosed in paired opening and closing tags. XML is stricter in this respect than HTML, which sometimes uses only opening tags. The following examples show paired opening and closing tags enclosing a value:

```
<interface>
  <name>eth0</name>
</interface>
```

The term *tag element* refers to the triple of 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.

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

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

NETCONF and SRC XML tag elements obey the XML convention that the tag element name indicates the kind of information enclosed by the tag element. For example, the name of the SRC XML **<interface>** tag element indicates that it contains information about an interface on the C Series Controller, whereas the name of the **<name>** tag element indicates that its contents specify the identifier.

When discussing tag elements in text, the convention is to use just the name of the opening tag to represent the complete tag element (opening tag, contents, and closing tag). For example, it usually refers to "the **<interface>** tag element" instead of "the **<interface><name>***name* **</name></interface>** tag element."

Document Type Definition

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

Advantages of Using the NETCONF and SRC XML APIs

The NETCONF and SRC XML APIs are programmatic interfaces. They fully document all options for every supported operational request and all elements in every configuration statement. The tag names clearly indicate the function of an element in an operational request or configuration statement.

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

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

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

This is the XML-tagged version:

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

When a client application needs to extract a specific value from formatted ASCII output, it must rely on the value's location, expressed either absolutely or with respect to labels

or values in adjacent fields. Suppose that the client application wants to extract the interface index. It can use a regular-expression matching utility to locate specific strings, but one difficulty is that the number of digits in the interface index is not necessarily predictable. The client application cannot simply read a certain number of characters after the **Interface index:** label, but must instead extract everything between the label and the subsequent label, which is:

```
, SNMP ifIndex
```

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

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

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

```
, Logical index
```

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

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

NETCONF Session Overview

Communication between the NETCONF server and a client application is session-based. The two parties explicitly establish a connection before exchanging data and close the connection when they are finished. The following list outlines the basic structure of a NETCONF session. For more specific information, see
"Controlling the NETCONF Session" on page 21.

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

CHAPTER 2

Using NETCONF and SRC XML Tag Elements

This chapter describes the syntactic and notational conventions used by the NETCONF server and client applications, including the mappings between statements and commands in the SRC command-line interface (CLI) and the tag elements in the SRC Extensible Markup Language (XML) application programming interface (API).

For more information about the syntax of CLI commands and configuration statements, see the *SRC PE CLI User Guide*. For information about specific configuration statements and operational mode commands, see the SRC documentation set.

This chapter includes the following topics:

Complying with XML and NETCONF Conventions

A client application must comply with XML and NETCONF conventions. Each request from the client application must be a *well-formed* XML document; that is, it must obey the structural rules defined in the NETCONF and SRC XML DTDs for the kind of information encoded in the request. The client application must emit tag elements in the required order and only in the legal contexts. Compliant applications are easier to maintain in the event of changes to the SRC software or NETCONF API.

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

Request and Response Tag Elements

A *request* tag element is one generated by a client application to request information about a C Series Controller'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 element. For information about the**<rpc>** tag element, see "Sending a Request to the NETCONF Server" on page 31.

A *response* tag element represents the NETCONF server's reply to a request tag element and occurs only within an **<rpc-reply>** tag element. For information about the **<rpc-reply>** tag element, see "Parsing the NETCONF Server Response" on page 34.

The following example represents an exchange in which a client application emits the **<get-interfaces>** request tag element and the NETCONF server returns the **<output>** response tag 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 NETCONF server. In practice, a client application does not need to include newline characters between tag elements, because the server automatically discards such white space. For further discussion, see "Spaces, Newline Characters, and Other White Space" on page 11.

For information about the **]]>]]> character sequence**, see "Generating Well-Formed XML Documents" on page 22. For information about the attributes in the opening **<rpc-reply>** tag, see "Parsing the NETCONF Server Response" on page 34. For information about the **xmlns** attribute in the opening **<output>** tag, see "Requesting Operational Information" on page 48.

Client Application	NETCONF Server
<rpc> <get-interfaces> <interface-name>eth0</interface-name> </get-interfaces> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <interface-information xmlns=" *URL*" > <!-- *children of <interface-information>* --> </interface-information> </rpc-reply>]]>]]>

Child Tag Elements of a Request Tag Element

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

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

In most cases, the client application can emit children that occur at the same level within a container tag element in any order. The important exception is a configuration element that has an *identifier tag element*, which distinguishes the configuration element from other elements of its type. The identifier tag element must be the first child tag element in the container tag element. Most frequently, the identifier tag element specifies the name of the configuration element and is called **<name>**. For more information, see "Mapping for Objects That Have an Identifier" on page 15.

Child Tag Elements of a Response Tag Element

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

The set of child tag elements that can occur in a response or within a container tag element is subject to change in later releases of the SRC XML API. Client applications must not rely on the presence or absence of a particular tag element in the NETCONF server's output, or on the ordering of child tag elements within a response tag element. For the most robust operation, include logic in the client application that handles the absence of expected tag elements or the presence of unexpected ones as gracefully as possible.

Spaces, Newline Characters, and Other White Space

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

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

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

XML Comments

Client applications and the NETCONF server can insert XML comments at any point between tag elements in the tag stream they generate, but not within tag elements. Client applications must handle comments in output from the NETCONF server gracefully but must not depend on their content. Client applications also cannot use comments to convey information to the NETCONF server, because the server automatically discards any comments it receives.

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

The following is an example of an XML comment:

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

Predefined Entity References

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

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

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

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

Table 3: Predefined Entity Reference Substitutions for Tag Content Values

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

Table 4 on page 13 summarizes the mapping between disallowed characters and predefined entity references for attribute values.

Table 4: Predefined Entity Reference Substitutions for Attribute Values

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

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

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

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

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

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

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

Mapping Commands to SRC XML Tag Elements

The SRC XML API defines tag-element equivalents for many commands in CLI operational mode. For example, the **<get-interfaces>** tag element corresponds to the **show interfaces** command.

For information about the available command equivalents in the current release of the SRC software, see the *SRC XML API Operational Reference*. For the mapping between commands and XML tag elements, see the table at the beginning of each chapter. For detailed information about a specific operation, see the appropriate section for the request tag element.

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

- Mapping for Command Options with Variable Values on page 14
- Mapping for Fixed-Form Command Options on page 14

Mapping for Command Options with Variable Values

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

The following example shows the XML tag elements for a CLI operational command that has variable-form options. In the **show interfaces** command, **eth0** is the name of the interface.

CLI Command	SRC XML Tags
show interfaces eth0	<rpc> <get-interfaces> <interface-name>eth0</interface-name> </get-interfaces> </rpc>

Mapping for Fixed-Form Command Options

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

The following example shows the XML tag elements for the **show disk status** command, which has a fixed-form option called **brief**.

CLI Command	SRC XML Tags
show disk status brief	<rpc> <get-disk-status> <brief/> </get-disk-status> </rpc>

Mapping Configuration Statements to SRC XML Tag Elements

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

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 XML tag element maps to both notational styles.

Mapping for Hierarchy Levels and Container Statements

The **<configuration>** tag element is the top-level XML container tag 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 XML container tag element that corresponds to a container statement almost always has the same name as the statement.

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

CLI Configuration Statements	SRC XML Tags
system login { **...child statements...** **}**	**<configuration>** **<system>** **<login>** **<!-- *tags for child statements* -->** **</login>** **</system>** **</configuration>**

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 SRC 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 SRC XML API differs from the CLI in its treatment of the identifier. Because the SRC 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 *SRC XML API Configuration Reference*.

NOTE: **The SRC software reserves the prefix sdx- for the identifiers of configuration groups defined within the sdx-defaults configuration group. User-defined identifiers cannot start with the string sdx-.**

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 SAE group configured at the **[edit shared sae group]** hierarchy level has an associated name (the identifier) and can have leaf statements for other characteristics, such as configuration, DHCP classification script, and subscriber classification script. For information about the XML mapping for leaf statements, see "Mapping for Single-Value and Fixed-Form Leaf Statements" on page 17, "Mapping for Leaf Statements with Multiple Values" on page 18, and "Using the Same Configuration Tag Elements in Requests and Responses" on page 19.

The following example shows the XML tag elements for configuration statements that define two users called **U1** and **U2**. Notice that the XML **<user-name>** tag element that encloses the identifier of each user does not have a counterpart in the CLI statements.

For complete information about changing C Series Controller configurations, see "Changing Configuration Information" on page 65.

CLI Configuration Statements	SRC XML Tags

system login {
user U1 {
class admin;
}
user U2 {
class admin;
}
}
	<configuration>
	<system>
	<login>
	<user>
	<user-name>U1</user-name> <!-- identifier -->
	<class>admin</class>
	</user>
	<user>
	<user-name>U2</user-name> <!-- identifier -->
	<class>admin</class>
	</user>
	</login>
	</system>
	</configuration>

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 XML tag element corresponding to a leaf statement is the same as the **keyword string.** The string between the opening and closing XML tags is the same as the ***value*** string.

The following example shows the XML tag elements for a leaf statement that has a keyword and a value: the **announcement** statement at the **[edit system login]** hierarchy level.

CLI Configuration Statements	SRC XML Tags

system login {
announcement " Authorized users only" ;
}
	<configuration>
	<system>
	<login>
	<announcement>Authorized users only
	</announcement>
	</login>
	</system>
	</configuration>

Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. The SRC XML API represents such statements with an empty tag.

The following example shows the XML tag elements for the **telnet** statement at the **[edit system services]** hierarchy level.

CLI Configuration Statements	SRC XML Tags

system services {
  telnet;
}
	<configuration>
	  <system>
	  <services>
	  <telnet/>
	  </services>
	  </system>
	</configuration>

Mapping for Leaf Statements with Multiple Values

Some 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 SRC XML API instead encloses each value in its own tag element. The following example shows the XML tag elements for a CLI statement with multiple user-defined values. The **domain-search** statement specifies two domains defined elsewhere in the configuration. For complete information about changing C Series Controller configurations, see "Changing Configuration Information" on page 65.

CLI Configuration Statements	SRC XML Tags

system {
  domain-search [jnpr.net juniper.net];
}
	<configuration>
	  <system>
	  <domain-search>jnpr.net</domain-search>
	  <domain-search>juniper.net</domain-search>
	  </system>
	</configuration>

The following example shows the 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	SRC XML Tags

system login class user-accounts {
  permissions [configure admin control];
}
	<configuration>
	 <system>
	  <login>
	  <class>
	  <name>user-accounts</name>
	  <permissions>configure</permissions>
	  <permissions>admin</permissions>
	  <permissions>control</permissions>
	  </class>
	  </login>
	 </system>
	</configuration>

Using the Same Configuration Tag Elements in Requests and Responses

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

The XML tag elements within the **<configuration>** tag element represent configuration hierarchy levels, configuration objects, and object characteristics, always ordered from higher to deeper levels of the hierarchy. When a client application loads a configuration, it can emit the same tag elements in the same order that the NETCONF server uses when returning configuration information. This consistent representation makes handling configuration information more straightforward. For instance, the client application can request the current configuration, store the NETCONF server's response in a local memory buffer, make changes or apply transformations to the buffered data, and submit the altered configuration as a change to the candidate configuration. Because the altered configuration is based on the NETCONF server's response, it is certain to be syntactically correct. For more information about changing C Series Controller configurations, see "Changing Configuration Information" on page 65.

Similarly, when a client application requests information about a configuration element (hierarchy level or configuration object), it uses the same tag elements that the NETCONF server will return in response. To represent the element, the client application sends a complete stream of tag elements from the top of the configuration hierarchy (represented by the **<configuration>** tag element) down to the requested element. The innermost tag element, which represents the level or object, is either empty or includes the identifier tag element only. The NETCONF server's response includes the same stream of parent tag elements, but the tag element for the requested configuration element contains all the tag elements that represent the element's characteristics or child levels. For more information, see "Requesting Configuration Information" on page 49.

The tag streams emitted by the NETCONF server and by a client application can differ in the use of white space, as described in "Spaces, Newline Characters, and Other White Space" on page 11.

CHAPTER 3

Controlling the NETCONF Session

This chapter explains how to start and terminate a session with the NETCONF server, and describes the Extensible Markup Language (XML) tag elements from the NETCONF application programming interface (API) that client applications and the NETCONF server use to coordinate information exchange during the session. It includes the following topics:

Client Application's Role in a NETCONF Session

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

1. Establishes a connection to the NETCONF server on the C Series Controller, as described in "Connecting to the NETCONF Server" on page 27.
2. Opens a NETCONF session, as described in "Starting the NETCONF Session" on page 28.
3. (Optional) Locks the candidate configuration, as described in "Locking the Candidate Configuration" on page 38. Locking the configuration prevents other users or applications from changing it at the same time.
4. Requests operational or configuration information, or changes configuration information, as described in "Requesting Information" on page 47 and "Changing Configuration Information" on page 65.
5. (Optional) Verifies the syntactic correctness of a configuration before attempting to commit it, as described in "Verifying a Configuration Before Committing It" on page 85.

6. Commits changes made to the configuration, as described in "Committing a Configuration" on page 85.
7. Unlocks the candidate configuration if it is locked, as described in "Unlocking the Candidate Configuration" on page 39.
8. Ends the NETCONF session and closes the connection to the C Series Controller, as described in "Ending a NETCONF Session and Closing the Connection" on page 41.

Establishing a NETCONF Session

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

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

For information about establishing a connection and NETCONF session, see the following sections:

- Generating Well-Formed XML Documents on page 22
- Prerequisites for Establishing a Connection on page 23
- Connecting to the NETCONF Server on page 27
- Starting the NETCONF Session on page 28

For an example of a complete NETCONF session, see "Example of a NETCONF Session" on page 42.

Generating Well-Formed XML Documents

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

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

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

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

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

Prerequisites for Establishing a Connection

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

Client Application Can Access SSH Software

The client application must be able to access the SSH software on the computer where it runs.

If the application uses the NETCONF Perl module provided by Juniper Networks, no further action is necessary. As part of the installation procedure for the Perl module, you install a prerequisites package that includes the necessary SSH software.

If the application does not use the NETCONF Perl module, 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.

Client Application Can Log In on C Series Controllers

The client application must be able to log in to each C Series Controller on which it establishes NETCONF sessions. The following instructions explain how to create a login account for the application. Alternatively, you can skip this section and enable authentication through RADIUS or TACACS+. For instructions, see *SRC PE Getting Started Guide*.

To determine if a login account exists, enter SRC command-line interface (CLI) configuration mode on the C Series Controller and issue the following commands:

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

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

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

1. Include the user statement at the **[edit system login]** hierarchy level. Specify a login class that has the permissions required for all actions to be performed by the application. You can also include the optional **full-name** and **uid** statements. For detailed information about creating user accounts, see *SRC PE Getting Started Guide*.

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

2. (Optional) Commit the configuration. Alternatively, you can wait until you have added the statements that satisfy all prerequisites (see "NETCONF Service over SSH Is Enabled" on page 26).

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

3. Repeat the preceding steps on each C Series Controller where the client application establishes NETCONF sessions.

Login Account Has Public/Private Key Pair or Password

For a client application to authenticate with the NETCONF server, the login account that you created in "Client Application Can Log In on C Series Controllers" on page 23 must have an SSH public/private key pair, a text-based password, or both. A public/private key pair is sufficient if the account is used only to connect to the NETCONF server through SSH. If the account is also used to access the C Series Controller in other ways (for login on the console, for example), it must have a text-based password. The password is also used (the SSH server prompts for it) if key-based authentication is configured but fails.

NOTE: You can skip this section if you have chosen to enable authentication through RADIUS or TACACS+, as described in *SRC PE Getting Started Guide*.

Follow the instructions in the appropriate section:

- Creating a Text-Based Password on page 24
- Creating a Public/Private Key Pair on page 25

Creating a Text-Based Password

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

1. Include either the **plain-text-password** or **encrypted-password** statement at the **[edit system login user *user-name* authentication]** hierarchy level. First, move to that hierarchy level:

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

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

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

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

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

2. (Optional) Commit the configuration. Alternatively, you can wait until you have added the statements that satisfy all prerequisites (see "NETCONF Service over SSH Is Enabled" on page 26).

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

3. Repeat the preceding steps on each C Series Controller where the client application establishes NETCONF sessions.

Creating a Public/Private Key Pair

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

1. Issue the **ssh-keygen** command in the standard command shell (not the SRC CLI) on the computer where the client application runs. By providing the appropriate arguments, you encode the public key with either RSA (supported by SSH versions 1 and 2) or the Digital Signature Algorithm (DSA, supported by SSH version 2). For more information, see the manual page for the **ssh-keygen** command. The SRC software uses SSH version 2 by default, but also supports version 1.

   ```
   % ssh-keygen options
   ```

2. Associate the public keys with the login account by including the **ssh-authorized-keys** statement at the **[edit system login user *user-name* authentication]** hierarchy level. The SRC software copies the public keys onto the C Series Controller:

   ```
   [edit system login user  user-name authentication]
   user@host# set ssh-authorized-keys [ssh-authorized-keys...]
   ```

   The **ssh-keygen** command by default stores each public key in a file in the **ssh** subdirectory of the user home directory; the filename depends on the encoding (DSA

or RSA) and SSH version. For more information about configuring SSH authentication, see *SRC PE Getting Started Guide*.

3. (Optional) Commit the configuration. Alternatively, you can wait until you have added the statements that satisfy all prerequisites (see "NETCONF Service over SSH Is Enabled" on page 26).

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

4. Repeat Steps 2 and 3 on each C Series Controller where the client application establishes NETCONF sessions.

Client Application Can Access the Keys or Password

The client application must be able to access the public/private keys or password you created in "Login Account Has Public/Private Key Pair or Password" on page 24 and provide it when the NETCONF server prompts for it.

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

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

NETCONF Service over SSH Is Enabled

The IETF draft titled *Using the NETCONF Configuration Protocol over Secure Shell (SSH)* requires that the NETCONF server by default provide SSH access to client machines over a devoted Transmission Control Protocol (TCP) port, to make it easy to identify and filter NETCONF traffic. The port for the SRC NETCONF server is 32000. In addition, you can enable client applications to access the NETCONF server over the default SSH port (22). (For more information about the IETF draft, see "Generating Well-Formed XML Documents" on page 22.)

Perform the following steps:

1. Include one or both of the following statements at the indicated hierarchy level:
   - To enable SSH access over the devoted port (32000) as specified by the IETF specification, include the **ssh** statement at the **[edit system services netconf]** hierarchy level:

```
[edit system login user user-name authentication]
user@host# top
[edit]
user@host# set system services netconf ssh
```

- To enable access over the default SSH port (22), include the **ssh** statement at the **[edit system services]** hierarchy level. This configuration also enables SSH access to the C Series Controller for all users and applications.

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

2. Commit the configuration:

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

3. Repeat the preceding steps on each C Series Controller where the client application establishes NETCONF sessions.

Connecting to the NETCONF Server

Before a client application can connect to the NETCONF server, you must satisfy the requirements described in "Prerequisites for Establishing a Connection" on page 23.

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

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

  ```
  ssh -p 32000 -s user@hostname netconf
  ```

  The **-p** option defines the port number on which the NETCONF server listens. This option can be omitted if you enabled access to SSH over the default port in "NETCONF Service over SSH Is Enabled" on page 26.

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

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

Starting the NETCONF Session

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

Exchanging <hello> Tag Elements

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

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

Each capability defined in the NETCONF specification is represented in a tag element by a uniform resource name (URN). Capabilities defined by individual vendors are represented by uniform resource identifiers (URIs), which can be URNs or URLs. The NETCONF API for SRC emits the following**<hello>** tag element (each**<capability>** tag element appears on three lines for legibility only):

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

(For information about the]]>]]> character sequence, see "Generating Well-Formed XML Documents" on page 22.)

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

- **urn:ietf:params:xml:ns:netconf:base:1.0**—The NETCONF server supports the basic NETCONF operations and tag elements defined in this namespace.
- **urn:ietf:params:xml:ns:netconf:capability:candidate:1.0**—The NETCONF server supports operations on a candidate configuration. For more information, see "Requesting

Information from the Candidate Configuration" on page 51 and "Committing Configurations" on page 85.

- **urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file**—The NETCONF server accepts configuration data stored in a file. It can retrieve files both from its local filesystem (indicated by the file option in the URN) and from remote machines by using Hypertext Transfer Protocol (HTTP) or FTP (indicated by the http and ftp options in the URN). For more information, see "Providing Configuration Data in a File" on page 67.
- http://xml.juniper.net/netconf/junos/1.0—The NETCONF server supports the operations defined in the SRC XML API for requesting and changing operational information (the tag elements in the *SRC XML API Operational Reference*).
- http://xml.juniper.net/netconf/junos/sdx/1.0—The NETCONF server supports the operations defined in the SRC XML API for requesting and changing operational information (the tag elements in the *SRC XML API Operational Reference*). The NETCONF server also supports operations for requesting or changing configuration information (the tag elements in the *SRC XML API Configuration Reference*).

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

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

Verifying Compatibility

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

```
<rpc>
    <get-system-info/>
</rpc>
]]>]]>
```

The NETCONF server returns the **<system-info>** tag element, which encloses the **<system-identification>** and **<software-version>** tag elements. (For information about the **<rpc-reply>** tag element, see "Parsing the NETCONF Server Response" on page 34.) Some tag elements appear on multiple lines, for legibility only:

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

```
            <software-version>SRC Release 2.0.0</software-version>
            <!- - other <system-identification> child tag elements - ->
        </system-identification>
        <!- - other <system-info> child tag elements - ->
    </system-info>
</rpc-reply>
]]>]]>
```

Normally, the version is the same for all SRC software components running on the C Series Controller. (We recommend this configuration for predictable performance.) The client application can determine the version of the SRC software components running on the NETCONF server by emitting the **<get-component-all>** tag element in an**<rpc>** tag element:

```
<rpc>
    <get-component-all/>
</rpc>
]]>]]>
```

The NETCONF server returns the **<sdx-component-list>** tag element, which encloses the **<sdx-component>** tag elements plus a **<version>** tag element for each installed SRC software component. (For information about the **<rpc-reply>** tag element, see "Parsing the NETCONF Server Response" on page 34.) The **<version>** tag element within the **<sdx-component>** tag element specifies the SRC release number (in the following example, 2.0 for SRC Release 2.0) and the build information. Some tag elements appear on multiple lines, for legibility only:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" \
                xmlns:sdx="http://xml.juniper.net/junos/sdx/1.0">
    <sdx-component-list>
        <sdx-component>
            <status>stopped</status>
            <version>Release: 2.0 Build: ACP.B.2.0.0.001</version>
            <name>acp</name>
        </sdx-component>
        <!- - other <sdx-component> tag elements - ->
        <sdx-component>
            <status>running</status>
            <version>Release: 2.0 Build: WEBADM.B.2.0.0.001</version>
            <name>webadm</name>
        </sdx-component>
    </sdx-component-list>
</rpc-reply>
]]>]]>
```

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

- Ignore differences in capabilities and SRC version, and do not alter the client application's behavior to accommodate the NETCONF server. A difference in SRC

versions does not necessarily make the server and client incompatible, so this is often a valid approach. Similarly, it is a valid approach if the capabilities that the client application does not support are operations that are always initiated by a client, such as validation of a configuration and confirmed commit. In that case, the client maintains compatibility by not initiating the operation.

- Alter standard behavior to be compatible with the NETCONF server. If the client application is running a later version of the SRC software, for example, it can choose to emit only NETCONF and SRC XML tag elements that represent the software features available in the NETCONF server's version of the software.
- End the NETCONF session and terminate the connection. This is appropriate if you decide that it is not practical to make the client application accommodate the SRC version or capabilities supported by the NETCONF server. For instructions, see "Ending a NETCONF Session and Closing the Connection" on page 41.

Exchanging Information with the NETCONF Server

The session continues when the client application sends a request to the NETCONF server. The NETCONF server does not emit any tag elements after session initialization except in response to the client application's requests. The following sections describe the exchange of tagged data:

Sending a Request to the NETCONF Server

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

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

Each request is enclosed in its own separate pair of opening **<rpc>** and closing **</rpc>** tags and must constitute a well-formed XML document by including only compliant and correctly ordered tag elements. For information about the **]]>]]>** character sequence, see "Generating Well-Formed XML Documents" on page 22. For an example of emitting an **<rpc>** tag element in the context of a complete NETCONF session, see "Example of a NETCONF Session" on page 42.

The NETCONF server ignores any newline characters, spaces, or other white space characters that occur between tag elements in the tag stream, but it preserves white space within tag elements. For more information, see "Spaces, Newline Characters, and Other White Space" on page 11.

See the following sections for further information:

- Request Classes on page 32
- Including Attributes in the Opening <rpc> Tag on page 34

Request Classes

A client application can make three classes of requests:

- Operational Requests on page 32
- Configuration Information Requests on page 32
- Configuration Change Requests on page 33

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

Operational Requests

Operational requests are requests for information about C Series Controller status, and correspond to the CLI operational mode commands listed in the SRC software command references. The SRC XML API defines a request tag element for many CLI commands. For example, the **<get-interfaces>** tag element corresponds to the **show interfaces** command, and the **<get-system-info>** tag element requests the same information as the **show system information** command.

The following sample request is for information about the interface called **eth0**:

```
<rpc>
    <get-interfaces>
        <interface-name>eth0</interface-name>
    </get-interfaces>
</rpc>
]]>]]>
```

For more information, see "Requesting Operational Information" on page 48. For information about the XML request tag elements available in the current SRC software release, see the *SRC XML API Operational Reference*.

Configuration Information Requests

Requests for configuration information are requests for information about the current configuration, either candidate or committed (the one currently in active use on the C Series Controller). The candidate and committed configurations diverge when there are uncommitted changes to the candidate configuration.

The NETCONF API defines the **<get-config>** tag element for retrieving configuration information. The SRC XML API defines a tag element for every CLI configuration statement described in the SRC software documentation set.

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

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

For more information, see "Requesting Configuration Information" on page 49. For a summary of the available configuration tag elements, see the *SRC XML API Configuration Reference*.

Configuration Change Requests

Configuration change requests are requests to change the candidate configuration, or to commit those changes to put them into active use on the C Series Controller. The NETCONF API defines the **<edit-config>** and **<copy-config>** tag elements for changes to the configuration. The SRC XML API defines a tag element for every CLI configuration statement described in the SRC software documentation set.

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

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

For more information, see "Changing Configuration Information" on page 65. For a summary of SRC XML configuration tag elements, see the *SRC XML API Configuration Reference*.

Including Attributes in the Opening <rpc> Tag

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

This feature can be used to associate requests and responses if the value assigned to an attribute by the client application is unique in each opening **<rpc>** tag. Because the NETCONF server echoes the attribute unchanged, it is simple to map the response to the initiating request. The NETCONF specification specifies the name **message-id** for this attribute.

Parsing the NETCONF Server Response

The NETCONF server encloses its response to each client request in a separate pair of opening **<rpc-reply>** and closing **</rpc-reply>** tags, each of which constitutes a well-formed XML document. In the opening **<rpc-reply>** tag, it includes the **xmlns** and **xmlns:sdx** attributes (the opening tag appears here on multiple lines for legibility only):

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

The **xmlns** attribute defines the namespace for enclosed tag elements that do not have the **sdx:** prefix on their names and that are not enclosed in a child container tag that has the **xmlns** attribute with a different value.

The **xmlns:sdx** attribute defines the namespace for enclosed tag elements that have the **sdx:** prefix on their names.

For information about the **]]>]]>** character sequence, see "Generating Well-Formed XML Documents" on page 22. For information about echoed attributes, see "Including Attributes in the Opening <rpc> Tag" on page 34.

Client applications must include code for parsing the stream of response tag elements coming from the NETCONF server, either processing them as they arrive or storing them until the response is complete. See the following sections for further information:

- NETCONF Server Response Classes on page 35
- Using a Standard API to Parse Response Tag Elements on page 36

NETCONF Server Response Classes

The NETCONF server returns three classes of responses:

- Operational Responses on page 35
- Configuration Information Responses on page 35
- Configuration Change Responses on page 36

Operational Responses

Operational responses are responses to requests for information about C Series Controller status. They correspond to the output from CLI operational commands as described in the SRC CLI command references.

The SRC XML API defines response tag elements for all defined operational request tag elements. For example, the NETCONF server returns the information requested by the **<get-system-info>** tag element in a response tag element called **<system-info>**.

The following sample response includes information about the C Series Controller. The namespace indicated by the **xmlns** attribute in the opening **<system-info>** tag is for system information. The opening tags appear on two lines here for legibility only:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"\
               xmlns:sdx="http://xml.juniper.net/junos/sdx/1.0">
    <system-info sdx:style="normal"\
               xmlns="http://xml.juniper.net/sdx/system-info"\
               xmlns:sdx="http://xml.juniper.net/sdx">
        <cpu xmlns="http://xml.juniper.net/sdx/cpu">
            <number>4</number>
            <model>Dual Core AMD Opteron(tm) Processor 265</model>
            <speed>1804.108 MHz</speed>
        </cpu>
        <!- - other data tag elements for <system-info> -  ->
    </system-info>
</rpc-reply>
]]>]]>
```

For more information about the **xmlns** attribute and the contents of operational response tag elements, see "Requesting Operational Information" on page 48. For a summary of operational response tag elements, see the *SRC XML API Operational Reference*.

Configuration Information Responses

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

The following sample response includes the information at the **[edit system login]** hierarchy level in the configuration hierarchy. For brevity, the sample shows only one user defined at this level. The opening **<rpc-reply>** tag appears on two lines for legibility only. For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

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

Configuration Change Responses

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

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

If the operation fails, the **<rpc-reply>** tag element instead encloses an **<rpc-error>** tag element that describes the cause of the failure. For information about handling errors, see "Handling an Error or Warning" on page 37.

For information about changing C Series Controller configuration, see "Changing Configuration Information" on page 65. For a summary of the available configuration tag elements, see the *SRC XML API Configuration Reference*.

Using a Standard API to Parse Response Tag Elements

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 document.

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

One potential drawback with DOM is that it always builds a hierarchy of tag elements, which can become very large. If a client application needs to handle only a 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.

Handling an Error or Warning

If the NETCONF server encounters an error condition, it emits an **<rpc-error>** tag element containing tag elements that describe the error:

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

- **<bad-element>** identifies the command or configuration statement that was being processed when the error or warning occurred. For a configuration statement, the **<error-path>** tag element enclosed in the **<rpc-error>** tag element specifies the statement's parent hierarchy level.
- **<error-message>** describes the error or warning in a natural-language text string.
- **<error-path>** specifies the path to the configuration hierarchy level at which the error or warning occurred, in the form of the CLI configuration mode banner.
- **<error-severity>** indicates the severity of the event that caused the NETCONF server to return the **<rpc-error>** tag element. The two possible values are **error** and **warning**.

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

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

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

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

Locking and Unlocking the Candidate Configuration

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

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

If an application is simply requesting configuration information and not changing it, locking the configuration is not required. The application can begin requesting information immediately, as described in "Requesting Configuration Information" on page 49. However, it is appropriate to lock the configuration if it is important that the information being returned not change during the session.

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

Locking the Candidate Configuration

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

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

Emitting these tag elements prevents other users or applications from changing the candidate configuration until the lock is released (equivalent to the CLI **configure exclusive** command). We recommend locking the configuration before you make changes, particularly on C Series Controllers where multiple users are authorized to change the configuration. A commit operation applies to all changes in the candidate configuration, not just those made by the user or application that requests the commit. Allowing multiple users or applications to make changes simultaneously can lead to unexpected results.

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

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

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

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

- Another user or application has already locked the candidate configuration. The error message reports the NETCONF session identifier of the user or application. If the client application has the necessary access privilege, it can terminate the session that holds the lock. For more information, see "Terminating Another NETCONF Session" on page 40.
- The candidate configuration already includes changes that have not yet been committed. To commit the changes, see "Committing a Configuration" on page 85. To discard uncommitted changes, see "Replacing the Candidate Configuration with the Running Configuration" on page 73.

Only one application can hold the lock on the candidate configuration at a time. Other users and applications can read the candidate configuration while it is locked. The lock persists until either the NETCONF session ends or the client application unlocks the configuration by emitting the **<unlock>** tag element, as described in "Unlocking the Candidate Configuration" on page 39.

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

Unlocking the Candidate Configuration

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

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

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

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

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

Terminating Another NETCONF Session

A client application's attempt to lock the candidate configuration can fail because another user or application already holds the lock, as mentioned in "Locking the Candidate Configuration" on page 38. In this case, the NETCONF server returns an error message that includes the username and process ID (PID) for the entity that holds the existing lock:

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

      exclusive
    </error-message>
  </rpc-error>
</rpc-reply>
]]>]]>
```

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

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

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

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

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

When a session is terminated, the NETCONF server that is servicing the session rolls back all uncommitted changes that have been made during the session.

The following example shows how to terminate a session:

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

Ending a NETCONF Session and Closing the Connection

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

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

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

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

For an example of the exchange of closing tag elements, see "Closing the NETCONF Session" on page 46.

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

Displaying CLI Output as XML Tag Elements

To display the output from a CLI command as NETCONF and SRC XML tag elements instead of as the default formatted ASCII, pipe the command to the **display xml** command. The tag elements that describe SRC configuration or operational data belong to the SRC XML API, which defines the content that can be retrieved and manipulated by the NETCONF API.

The following example shows the output from the **show system information** command issued on a C Series Controller (the opening **<system-info>** tag appears on multiple lines for legibility only):

```
user@host> show system information |display xml
  <?xml version="1.0" encoding="utf-8"?>
      <system-info sdx:style="normal"\
                  xmlns="http://xml.juniper.net/sdx/system-info"\
                  xmlns:sdx="http://xml.juniper.net/sdx">
          <cpu xmlns="http://xml.juniper.net/sdx/cpu">
              <number>4</number>
              <model>Dual Core AMD Opteron(tm) Processor 265</model>
              <speed>1804.108 MHz</speed>
          </cpu>
          <!- - other child tags of <system-info> - ->
      </system-info>
```

Example of a NETCONF Session

This section describes the sequence of tag elements in a sample NETCONF session. The client application begins by establishing a connection to a NETCONF server. See the following sections:

Exchanging Initialization Tag Elements

After the client application establishes a connection to a NETCONF server, the two exchange **<hello>** tag elements, as shown in the following example. For legibility, the example places the client application's **<hello>** tag element below the NETCONF server's. The two parties can actually emit their **<hello>** tag elements at the same time. For information about the **]]>]]>** character sequence used in this and the following examples, see "Generating Well-Formed XML Documents" on page 22. For a detailed discussion of the**<hello>** tag element, see "Exchanging <hello> Tag Elements" on page 28.

NETCONF Server	Client Application

```
<hello>
 <capabilities>
  <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>
  <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?candidate:1.0</capability>
  <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
  <capability>http://xml.juniper.net/netconf/junos/sdx/1.0</capability>
 </capabilities>
 <session-id>3911</session-id>
</hello>
]]>]]>
```

NETCONF Server	Client Application
	<hello> <capabilities> <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability> <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</ca <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?candidate <capability>http://xml.juniper.net/netconf/junos/1.0</capability> <capability>http://xml.juniper.net/netconf/junos/sdx/1.0</capability> </capabilities> </hello>]]>]]>

Sending an Operational Request

The client application now emits the **<get-system-info>** tag element to request information about the C Series Controller's chassis hardware. The NETCONF server returns the requested information in the **<system-info>** tag element.

Client Application	NETCONF Server
<rpc> <get-system-info/> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <system-info> <cpu> <!-- *other tags for <cpu>* --> </cpu> <!-- *other tags for <system-info>* --> </system-info> </rpc-reply>]]>]]>

Locking the Configuration

The client application then prepares to incorporate a change into the candidate configuration by emitting the **<lock>** tag to prevent any other users or applications from altering the candidate configuration at the same time. To confirm that the candidate configuration is locked, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag

element. For more information about locking the configuration, see "Locking the Candidate Configuration" on page 38.

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

Changing the Configuration

The client application now emits tag elements to create a new login class called **network-mgmt** at the **[edit system login class]** hierarchy level in the candidate configuration. To confirm that it incorporated the changes, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag element. (Understanding the meaning of these tag elements is not necessary for the purposes of this example, but for information about them, see "Changing Configuration Information" on page 65.)

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <config> <configuration> <system> <login> <class> <name>network-mgmt</name> <permissions>configure</permissions> <permissions>snmp</permissions> <permissions>system</permissions> </class> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	

Client Application	NETCONF Server
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Committing the Configuration

The client application commits the candidate configuration. To confirm that it committed the candidate configuration, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag element. For more information about the commit operation, see "Committing a Configuration" on page 85.

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

Unlocking the Configuration

The client application unlocks (and by implication closes) the candidate configuration. To confirm that it unlocked the candidate configuration, the NETCONF server returns an **<ok/>** tag in an **<rpc-reply>** tag element. For more information about unlocking the configuration, see "Unlocking the Candidate Configuration" on page 39.

Client Application	NETCONF Server
<rpc> <unlock> <target> <candidate/> </target> </unlock> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Closing the NETCONF Session

The client application closes the NETCONF session. For more information about closing the session, see "Ending a NETCONF Session and Closing the Connection" on page 41.

Client Application	NETCONF Server
<rpc> <close-session/> </rpc>]]>]]>	
	<rpc-reply xmlns=" *URN* " xmlns:sdx=" *URL* " > <ok/> </rpc-reply>]]>]]>

CHAPTER 4

Requesting Information

This chapter explains how to use the SRC Extensible Markup Language (XML) and NETCONF application programming interfaces (APIs) to request information about C Series Controller status and the current configuration.

The tag elements for operational requests are defined in the SRC XML API and correspond to command-line interface (CLI) operational commands, which are described in the SRC software command references. There is a request tag element for many commands in the CLI **show** family of commands.

The tag element for configuration requests is the NETCONF **<get-config>** tag element. It corresponds to the CLI configuration mode **show** command, which is described in the *SRC PE CLI User Guide*. The SRC XML tag elements that make up the content of both requests and the NETCONF server's responses correspond to CLI configuration statements, which are described in the SRC software documentation set.

In addition to information about the current configuration, client applications can request other configuration-related information.

This chapter includes the following topics:

Request Procedure Overview

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

1. Establishes a connection to the NETCONF server on the C Series Controller, as described in "Connecting to the NETCONF Server" on page 27.
2. Opens a NETCONF session, as described in "Starting the NETCONF Session" on page 28.
3. If making configuration requests, optionally locks the candidate configuration, as described in "Locking the Candidate Configuration" on page 38.

4. Makes any number of requests one at a time, freely intermingling operational and configuration requests. See "Requesting Operational Information" on page 48 and "Requesting Configuration Information" on page 49.

   The application can also intermix requests with configuration changes, which are described in "Changing Configuration Information" on page 65.

5. Accepts the tag stream emitted by the NETCONF server in response to each request and extracts its content, as described in "Parsing the NETCONF Server Response" on page 34.

6. Unlocks the candidate configuration if it is locked, as described in "Unlocking the Candidate Configuration" on page 39. Other users and applications cannot change the configuration while it remains locked.

7. Ends the NETCONF session and closes the connection to the C Series Controller, as described in "Ending a NETCONF Session and Closing the Connection" on page 41.

Requesting Operational Information

To request information about the current status of a C Series Controller, a client application emits the specific tag element from the SRC XML API that returns the desired information. For example, the **<get-interfaces>** tag element corresponds to the **show interfaces** command, and the **<get-system-info>** tag element requests the same information as the **show system information** command.

For complete information about the operational request tag elements available in the current SRC software release, see the chapters in the *SRC XML API Operational Reference* that are titled " Mapping Between Operational Tag Elements and CLI Commands" and " Summary of Operational Request Tag Elements."

The application encloses the request tag element in an **<rpc>** tag element. The syntax depends on whether the corresponding CLI command has any options:

```
<rpc>
   <!- - If the command does not have options - ->
      <operational-request/>

   <!- - If the command has options - ->
      <operational-request>
        <!- - tag elements representing the options - ->
      </operational-request>
</rpc>
]]>]]>
```

The NETCONF server encloses its response in a specific tag element that matches the request tag element, enclosed in an **<rpc-reply>** tag element.

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <operational-response xmlns="URL-for-DTD">
        <!- -  SRC  XML tag elements for the requested information - ->
    </operational-response>
</rpc-reply>
]]>]]>
```


The opening tag for each operational response includes the **xmlns** attribute to define the XML namespace for the enclosed tag elements that do not have a prefix (such as **sdx:**) in their names. The namespace indicates which XML document type definition (DTD) defines the set of tag elements in the response. The SRC XML API defines separate DTDs for operational responses from different software components. For instance, the DTD for software component information is called **SdxComponentList.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/sdx/category

where ***category*** specifies the DTD for the top-level tag.

For example, http://xml.juniper.net/sdx/sdx-component-list would be the namespace for the SdxComponentList DTD.

The *SRC XML API Operational Reference* includes the text of the SRC XML DTDs for operational responses.

Parsing the <output> Tag Element

If the SRC XML API does not define a response tag element for the type of output requested by a client application, the NETCONF server encloses its response in an **<output>** tag element. The tag element's contents are usually one or more lines of formatted ASCII output like that displayed by the CLI on the computer screen.

NOTE: The content and formatting of data within an <output> tag element are subject to change, so client applications must not depend on them. Future versions of the SRC XML API will define specific response tag elements (instead of <output> tag elements) for more commands. Client applications that rely on the content of <output> tag elements will not be able to interpret the output from future versions of the SRC XML API.

Requesting Configuration Information

To request information about a configuration on a C Series Controller, a client application encloses the **<get-config>**, **<source>**, and **<filter>** tag elements in an **<rpc>** tag element. By including the appropriate child tag element in the **<source>** tag element, the client application requests information from either the candidate or active configuration. By including the appropriate child tag elements in the **<filter>** tag element, the application can request the entire configuration or portions of it:

```
<rpc>
   <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
```

```
        </source>
        <filter type="subtree">
            <!- - tag elements representing the configuration elements to
return - ->
        </filter>
   </get-config>
</rpc>
]]>]]>
```

The **type="subtree"** attribute in the opening **<filter>** tag indicates that the client application is using XML tag elements to represent the configuration elements about which it is requesting information. For information about the syntax used within the **<filter>** tag element to represent elements, see "Specifying the Scope of Configuration Information to Return" on page 51.

NOTE: If the client application locks the candidate configuration before making requests, it needs to unlock the configuration after making its read requests. Other users and applications cannot change the configuration while it remains locked. For more information, see "Locking and Unlocking the Candidate Configuration" on page 38.

The NETCONF server encloses its reply in **<configuration>**, **<data>**, and **<rpc-reply>** tag elements. It includes attributes in the opening **<configuration>** tag that indicate the XML namespace for the enclosed tag elements and when the configuration was last changed or committed. For information about the attributes, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - SRC XML tag elements representing configuration elements -
 ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

If an XML tag element is returned within an **<undocumented>** tag element, the corresponding configuration element is not documented in the SRC software configuration guides or officially supported by Juniper Networks. Most often, the enclosed element is used for debugging only by Juniper Networks personnel. In a smaller number of cases, the element is no longer supported or has been moved to another area of the configuration hierarchy, but appears in the current location for backward compatibility.

Client applications can also request other configuration-related information.

The following sections describe how a client application specifies the source and scope of configuration information returned by the NETCONF server:

- Requesting Information from the Candidate Configuration on page 51
- Specifying the Scope of Configuration Information to Return on page 51

Requesting Information from the Candidate Configuration

To request information from the candidate configuration, a client application includes the **<source>** tag element and **<candidate/>** tag in **<rpc>** and **<get-config>** tag elements:

```
<rpc>
    <get-config>
      <source>
        <candidate/>
      </source>
       <filter>
            <!- - tag elements representing the configuration elements to
return - ->
       </filter>
    </get-config>
</rpc>
]]>]]>
```

NOTE: If requesting the entire configuration, the application omits the **<filter>** tag element. For information about the **<filter>** tag element, see "Specifying the Scope of Configuration Information to Return" on page 51.

The NETCONF server encloses its reply in **<configuration>**, **<data>**, and **<rpc-reply>** tag elements. In the opening **<configuration>** tag, it includes the **xmlns** attribute to specify the namespace for the enclosed tag elements.

When returning information from the candidate configuration, the NETCONF server also includes attributes that indicate when the configuration last changed (they appear on multiple lines here only for legibility):

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <data>
        <configuration xmlns="URL" sdx:changed-seconds=seconds" \
                sdx:changed-localtime="YYYY-MM-DD hh:mm:ss TZ">
            <!- - SRC XML tag elements representing the configuration - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

sdx:changed-localtime represents the time of the last change as the date and time in the C Series Controller's local time zone.

sdx:changed-seconds represents the time of the last change as the number of seconds since midnight on 1 January 1970.

Specifying the Scope of Configuration Information to Return

By including the appropriate child tag elements in the **<filter>** tag element within the **<rpc>** and **<get-config>** tag elements, a client application can request the entire configuration or portions of it:

```
<rpc>
    <get-config>
        <source>
            <candidate/>
        </source>
        <filter>
            <!- - tag elements representing the configuration elements to
return - ->
        </filter>
    </get-config>
</rpc>
]]>]]>
```

For information about requesting different amounts of configuration information, see the following sections:

- Requesting the Complete Configuration on page 52
- Requesting a Hierarchy Level or Container Object Without an Identifier on page 53
- Requesting All Configuration Objects of a Specified Type on page 54
- Requesting Identifiers for Configuration Objects of a Specified Type on page 56
- Requesting One Configuration Object on page 58
- Requesting Specific Child Tags for a Configuration Object on page 60
- Requesting Multiple Configuration Elements Simultaneously on page 62

Requesting the Complete Configuration

To request the entire candidate configuration, a client application encloses **<get-config>** and **<source>** tag elements and the **<candidate/>** tag in an **<rpc>** tag element:

```
<rpc>
    <get-config>
      <source>
        <candidate/>
      </source>
    </get-config>
</rpc>
]]>]]>
```

The NETCONF server encloses its reply in **<configuration>**, **<data>**, and **<rpc-reply>** tag elements. For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN "xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - SRC XML tag elements representing the configuration - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

Requesting a Hierarchy Level or Container Object Without an Identifier

To request complete information about all child configuration elements at a hierarchy level or in a container object that does not have an identifier, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels in the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level of the level or container object, which is represented by an empty tag. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
    <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
        </source>
        <filter type="subtree">
            <configuration>
                <!- - opening tags for each parent of the requested level -
->
                <level-or-container/>
                <!- - closing tags for each parent of the requested level -
->
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see "Requesting Information from the Candidate Configuration" on page 51.

The NETCONF server returns the requested section of the configuration in **<data>** and **<rpc-reply>** tag elements. For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - opening tags for each parent of the level - ->
                <level-or-container>
                    <!- - child tag elements of the level or container - ->
                </level-or-container>
            <!- - closing tags for each parent of the level - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see "Requesting Multiple Configuration Elements Simultaneously" on page 62.

The following example shows how to request the contents of the **[edit system login]** hierarchy level in the candidate configuration.

Client Application	NETCONF Server
<rpc> <get-config> <source> <candidate/> </source> <filter> <configuration> <system> <login/> </system> </configuration> </filter> </get-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <data> <configuration xmlns="*URL*" \ sdx:changed-seconds="seconds" \ sdx:changed-localtime="timestamp"> <system> <login> <user> <name>barbara</name> <full-name>Barbara Anderson</full-name> <class>super-user</class> <uid>632</uid> </user> <!-- *other child tag elements of <login>* --> </login> </system> </configuration> </data> </rpc-reply>]]>]]>

Requesting All Configuration Objects of a Specified Type

To request complete information about all configuration objects of a specified type in a hierarchy level, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels in the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object type. An empty tag represents the requested object type. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
    <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
        </source>
```

```
        <filter type="subtree">
            <configuration>
                <!- - opening tags for each parent of the requested object
type - ->
                    <object-type/>
                <!- - closing tags for each parent of the requested object
type - ->
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see "Requesting Information from the Candidate Configuration" on page 51.

This type of request is useful when the object's parent hierarchy level has more than one type of child object. If the requested object is the only child type that can occur in its parent hierarchy level, then this type of request yields the same output as a request for the complete parent hierarchy, which is described in "Requesting a Hierarchy Level or Container Object Without an Identifier" on page 53.

The NETCONF server returns the requested objects in **<data>** and **<rpc-reply>** tag elements. For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN "xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - opening tags for each parent of the object type - ->
                <first-object>
                    <!- - child tag elements for the first object - ->
                </first-object>
                <second-object>
                    <!- - child tag elements for the second object - ->
                </second-object>
                <!- - additional instances of the object - ->
            <!- - closing tags for each parent of the object type - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see "Requesting Multiple Configuration Elements Simultaneously" on page 62.

The following example shows how to request complete information about all **radius-server** objects at the **[edit system]** hierarchy level in the candidate configuration.

Client Application	NETCONF Server

```
<rpc>
 <get-config>
  <source>
   <candidate/>
  </source>
  <filter>
   <configuration>
    <system>
     <radius-server/>
    </system>
   </configuration>
  </filter>
 </get-config>
</rpc>
]]>]]>
```

```
<rpc-reply xmlns="URN" xmlns:sdx="URL" >
 <data>
  <configuration xmlns="URL" \
    sdx:changed-seconds="seconds" \
    sdx:changed-localtime="timestamp">
   <system>
    <radius-server>
     <address>10.25.34.166</address>
     <port>1812</port>
     <secret>$9$Pf3900REcr/9t...</secret>
     <timeout>5</timeout>
     <retry>3</retry>
    </radius-server>
    <radius-server>
     <address>10.25.6.204</address>
     <port>1812</port>
     <secret>$9$K5Kvxd2gJZUi-d...</secret>
     <timeout>5</timeout>
     <retry>3</retry>
    </radius-server>
   </system>
  </configuration>
 </data>
</rpc-reply>
]]>]]>
```

Requesting Identifiers for Configuration Objects of a Specified Type

To request output that shows only the identifier for each configuration object of a specific type in a hierarchy, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object type. The object type is represented by its container tag element enclosing an empty **<name/>** tag. (The **<name>** tag element can always be used, even if the actual identifier

tag element has a different name. The actual name is also valid.) The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
    <get-config>
        <source>
            <!-tag specifying the source configuration - -> -
        </source>
        <filter type="subtree">
            <configuration>
                <!- - opening tags for each parent of the object type - ->
                <object-type>
                    <name/>
                </object-type>
                <!- - closing tags for each parent of the object type - ->
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see "Requesting Information from the Candidate Configuration" on page 51.

NOTE: It is not possible to request only identifiers for object types that have multiple identifiers. However, for many such objects the identifiers are the only child tag elements, so requesting complete information yields the same output as requesting only identifiers. For instructions, see "Requesting All Configuration Objects of a Specified Type" on page 54.

The NETCONF server returns the requested objects in **<data>** and **<rpc-reply>** tag elements (here, objects for which the identifier tag element is called **<name>**). For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - opening tags for each parent of the object type - ->
                <first-object>
                    <name>identifier-for-first-object</name>
                </first-object>
                 <second-object>
                    <name>identifier-for-second-object</name>
                </second-object>
                <!- - additional objects - ->
            <!- - closing tags for each parent of the object type - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element.

For more information, see "Requesting Multiple Configuration Elements Simultaneously" on page 62.

The following example shows how to request the identifier for each file configured at the **[edit system syslog file]** hierarchy level in the candidate configuration.

Client Application	NETCONF Server

```
<rpc>
 <get-config>
  <source>
   <candidate/>
  </source>
  <filter>
   <configuration>
    <system>
     <syslog>
      <file/>
     </syslog>
    </system>
   </configuration>
  </filter>
 </get-config>
</rpc>
]]>]]>
```

```
<rpc-reply xmlns="URN" xmlns:sdx="URL" >
 <data>
  <configuration xmlns="URL" \
    sdx:changed-seconds="seconds" \
    sdx:changed-localtime="timestamp">
   <system>
    <syslog>
     <file>
      <name>file1</name>
      <name>file2</name>
     </file>
    </syslog>
   </system>
  </configuration>
 </data>
</rpc-reply>
]]>]]>
```

Requesting One Configuration Object

To request complete information about a specific configuration object, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object.

To represent the requested object, the application emits its container tag element and each of its identifier tag elements, complete with identifier value. 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 *SRC XML API Configuration Reference*. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
    <get-config>
        <source>
            <!- -tag specifying the source configuration - ->
        </source>
        <filter type="subtree">
            <configuration>
                <!- - opening tags for each parent of the object - ->
                <object>
                   <name>identifier</name>
                </object>
                <!- - closing tags for each parent of the object - ->
            </configuration>
        </filter >
    </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see "Requesting Information from the Candidate Configuration" on page 51.

The NETCONF server returns the requested object in **<data>** and **<rpc-reply>** tag elements (here, an object for which the identifier tag element is called <name>). For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN"xmlns:sdx ="URL">
    <data>
        <configuration attributes>
            <!- - opening tags for each parent of the object - ->
                <object>
                    <name>identifier</name>
                    <!- - other child tag elements of the object - ->
                </object>
            <!- - closing tags for each parent of the object - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see "Requesting Multiple Configuration Elements Simultaneously" on page 62.

The following example shows how to request the contents of the user called **barbara**, which is at the **[edit system login user]** hierarchy level in the candidate configuration. To

specify the desired object, the client application emits the **<name>barbara</name>** identifier tag element as the innermost tag element.

Client Application	NETCONF Server
<rpc> <get-config> <source> <candidate/> </source> <filter> <configuration> <system> <login> <user> <name>barbara</name> </user> </login> </system> </configuration> </filter> </get-config> </rpc>]]>]]>	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <data> <configuration xmlns="*URL*" \ sdx:changed-seconds="seconds" \ sdx:changed-localtime="timestamp"> <system> <login> <user> <name>barbara</name> <full-name>Barbara Anderson</full-name> <class>super-user</class> <uid>632</uid> </user> <!-- *other child tag elements of <login>* --> </login> </system> </configuration> </data> </rpc-reply>]]>]]>

Requesting Specific Child Tags for a Configuration Object

To request specific child tag elements for a specific configuration object, a client application emits a **<filter>** tag element that encloses the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to the immediate parent level for the object. To represent the requested object, the application emits its container tag element and identifier tag element. For objects with a single identifier, the **<name>** tag element can always be used, even if the actual identifier tag element has a different name. The actual name is also valid. For objects with multiple identifiers, the actual names of the identifier tag elements

must be used. To represent the child tag elements to return, it emits each one as an empty tag. The entire request is enclosed in an **<rpc>** tag element:

```
<rpc>
    <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
        </source>
        <filter type="subtree">
                <configuration>
                <!- - opening tags for each parent of the object - ->
                   <object>
                       <name>identifier</name>
                       <first-child/>
                       <second-child/>
                        <!- - empty tag for each additional child to return
- ->
                   </object>
                <!- - closing tags for each parent of the object - ->
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>
```

For information about the **<source>** tag element, see "Requesting Information from the Candidate Configuration" on page 51.

The NETCONF server returns the requested children of the object in **<data>** and **<rpc-reply>** tag elements (here, an object for which the identifier tag element is called **<name>**). For information about the attributes in the opening **<configuration>** tag, see "Requesting Information from the Candidate Configuration" on page 51.

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <data>
        <configuration attributes>
            <!- - opening tags for each parent of the object - ->
                <object>
                    <name>identifier</name>
                    <!- - requested child tag elements - ->
                </object>
            <!- - closing tags for each parent of the object - ->
        </configuration>
    </data>
</rpc-reply>
]]>]]>
```

The application can also request additional configuration elements of the same or other types by including the appropriate tag elements in the same **<get-config>** tag element. For more information, see "Requesting Multiple Configuration Elements Simultaneously" on page 62.

The following example shows how to request only the address of the next-hop router for the **192.168.5.0/24** route at the **[edit routing-options static route]** hierarchy level in the candidate configuration.

Client Application	NETCONF Server
<rpc> <get-config> <source> <candidate/> </source> <filter> <configuration> <routing-options> <static> <route> <destination>192.168.5.0/24</destination> <next-hop/> </route> </static> </routing-options> </configuration> </filter> </get-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <data> <configuration xmlns="*URL*" \ sdx:changed-seconds="seconds" \ sdx:changed-localtime="timestamp"> <routing-options> <static> <route> <destination>192.168.5.0/24</destination> <next-hop>192.168.71.254</next-hop> </route> </static> </routing-options> </configuration> </data> </rpc-reply>]]>]]>

Requesting Multiple Configuration Elements Simultaneously

Within a **<get-config>** tag element, a client application can request multiple configuration elements of the same type or different types. The request includes only one **<filter>** and **<configuration>** tag element. (The NETCONF server returns an error if there is more than one of each.)

If two requested objects have the same parent hierarchy level, the client can either include both requests within one parent tag element, or repeat the parent tag element for each request. For example, at the **[edit system]** hierarchy level the client can request the list

of configured services and the identifier tag element for RADIUS servers in either of the following two ways:

```
<!- - both requests in one <system> tag element - ->
<rpc>
    <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
        </source>
        <filter type="subtree">
            <configuration>
                <system>
                    <services/>
                    <radius-server>
                        <name/>
                    </radius-server>
                </system>
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>

<!- - separate <system> tag element for each element - ->
<rpc>
    <get-config>
        <source>
            <!- - tag specifying the source configuration - ->
        </source>
        <filter type="subtree">
            <configuration>
                <system>
                    <services/>
                </system>
                <system>
                    <radius-server>
                        <name/>
                    </radius-server>
                </system>
            </configuration>
        </filter>
    </get-config>
</rpc>
]]>]]>
```

The client can combine requests for any of the following types of information:

- Requesting a Hierarchy Level or Container Object Without an Identifier on page 53
- Requesting All Configuration Objects of a Specified Type on page 54
- Requesting Identifiers for Configuration Objects of a Specified Type on page 56
- Requesting One Configuration Object on page 58
- Requesting Specific Child Tags for a Configuration Object on page 60

CHAPTER 5

Changing Configuration Information

This chapter explains how to use the SRC Extensible Markup Language (XML) and NETCONF application programming interfaces (APIs) to change C Series Controller configuration. The NETCONF **<edit-config>** tag element corresponds to configuration mode commands in the SRC command-line interface (CLI), which are described in the *SRC PE CLI User Guide*. The SRC XML tag elements described here correspond to configuration statements, which are described in the SRC software documentation set.

This chapter includes the following topics:

Configuration Changes Overview

To change configuration information, the client application performs the procedures described in the indicated sections:

1. Establishes a connection to the NETCONF server on the C Series Controller, as described in "Connecting to the NETCONF Server" on page 27.
2. Opens a NETCONF session, as described in "Starting the NETCONF Session" on page 28.
3. (Optional) Locks the candidate configuration, as described in "Locking the Candidate Configuration" on page 38. Locking the configuration prevents other users or applications from changing it at the same time.
4. Encloses the **<edit-config>** and **<target>** tag elements and the **<candidate/>** tag in an **<rpc>** tag element. By including various child tag elements, the application can provide the configuration data either in a file or as a directly loaded tag stream, and can completely replace the existing configuration or specify the manner in which the NETCONF server loads the data into the existing candidate or copy. See "Changing the Candidate Configuration" on page 66.

5. Accepts the tag stream emitted by the NETCONF server in response to each request and extracts its content, as described in "Parsing the NETCONF Server Response" on page 34.
6. (Optional) Verifies the syntactic correctness of a configuration before attempting to commit it, as described in "Verifying a Configuration Before Committing It" on page 85.
7. Commits changes made to the configuration, as described in "Committing Configurations" on page 85.
8. Unlocks the candidate configuration if it is locked, as described in "Unlocking the Candidate Configuration" on page 39. Other users or applications cannot change the configuration while it remains locked.
9. Ends the NETCONF session and closes the connection to the C Series Controller, as described in "Ending a NETCONF Session and Closing the Connection" on page 41.

Changing the Candidate Configuration

To change the candidate configuration on a C Series Controller, a client application encloses the **<edit-config>** and **<target>** tag elements and the **<candidate/>** tag in the **<rpc>** tag element:

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

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

        <default-operation>(merge | none | replace)</default-operation>
    </edit-config>
</rpc>
]]>]]>
```

The other child tag elements in the preceding syntax statement specify additional parameters, and are described in the indicated sections:

- The **<url>** or **<config>** tag element defines the new configuration data to incorporate into the candidate. See "Defining the New Configuration Data" on page 67.
- The **<default-operation>** tag element specifies the default manner in which the NETCONF server incorporates new configuration data into the candidate configuration. See "Setting the Default Mode for Incorporating New Configuration Data" on page 70.

The NETCONF server confirms that it incorporated the configuration data by returning the **<ok/>** tag in the **<rpc-reply>** tag element:

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

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

Regardless of the value provided, the NETCONF server for the SRC software performs a basic syntax check on the configuration data in the **<edit-config>** tag element. It performs a complete syntactic and semantic validation in response to the **<commit>** tag element (that is, when the configuration is committed),but not in response to the **<edit-config>** tag element. For information about the **<commit>** tag element, see "Committing Configurations" on page 85.

The client application can also include the operation attribute in the opening tag for a configuration element to specify the manner in which to incorporate the element, which can differ from the manner specified by the **<default-operation>** tag element. See "Changing Individual Configuration Elements" on page 73.

Defining the New Configuration Data

A client application can use one of two ways to define the new data to incorporate into the candidate configuration:

- Providing Configuration Data in a File on page 67
- Providing Configuration Data as a Data Stream on page 68

Providing Configuration Data in a File

To provide the new configuration data in a file, a client application emits the **<url>** and **<edit-config>** tag elements in an**<rpc>** tag element:

```
<rpc>
    <edit-config>
        <target>
            <candidate/>
        </target>
        <url>
          <!- - location of file containing configuration data - ->
        </url>
        <!- - other child tag elements of the <edit-config> tag element - ->
    </edit-config>
</rpc>
]]>]]>
```

Before loading the file, the client application or an administrator saves XML tag elements as the contents of the file. The file includes the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to each element to change. The notation is the same as that used to request

configuration information, as described in "Requesting Information" on page 47. For more detailed information about the XML representation of configuration statements, see "Mapping Configuration Statements to SRC XML Tag Elements" on page 15.

The file named by the **<url>** tag element can be specified as a local file path, an FTP location, or a Hypertext Transfer Protocol (HTTP) URL:

- A local filename can have one of the following forms:
  - ***/path/filename***—File on a mounted file system, either on the local flash disk or on the hard disk.
  - **a:*filename*** or **a:*path/filename***—File on the local drive. The default path is / (the root-level directory). The removable media can be in MS-DOS or UNIX (UFS) format.
- A filename on an FTP server has the following form:

  ftp://*username:password@hostname/path/filename*
- A filename on an HTTP server has the following form:

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

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**, as in **ftp://*username:password@hostname/%2Fpath/filename***.

The following example shows how to incorporate configuration data stored in the file **/var/configs/user-accounts** on the FTP server called **cfg-server.mycompany.com**.

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <url>ftp://admin:AdminPwd@cfg-server.mycompany.com/var/configs/user-accounts</url> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Providing Configuration Data as a Data Stream

To provide configuration data as a data stream, a client application emits **<rpc>**, **<edit-config>**, and **<config>** tag elements. To specify the configuration elements to change, the application emits the tag elements representing all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to each element to change. The notation is the same as that used to request configuration

information, as described in "Requesting Information" on page 47. For more detailed information about the mappings between configuration elements and XML tag elements, see "Mapping Configuration Statements to SRC XML Tag Elements" on page 15.

```
<rpc>
    <edit-config>
        <target>
            <candidate/>
        </target>
        <config>
            <configuration>
               <!- - tag elements representing the configuration changes - ->

            </configuration>
        </config>
        <!- - other child tag elements of the <edit-config> tag element - ->
    </edit-config>
</rpc>
]]>]]>
```

The following example shows how to provide new configuration data for the **messages system log file** in a data stream:

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <config> <configuration> <system> <syslog> <file> <name>messages</name> <contents> <name>any</name> <warning/> </contents> <contents> <name>authorization</name> <info/> </contents> </file> </syslog> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Setting the Default Mode for Incorporating New Configuration Data

By default, the NETCONF server *merges* new configuration data into the candidate configuration, according to the following rules:

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

Merge mode applies to all elements in the new configuration that do not have the **operation** attribute in their opening container tag to specify a different mode. (For information about the operation attribute, see "Changing Individual Configuration Elements" on page 73.)

Merge mode is the default mode for incorporating new configuration data (it is used when a client application does not specify a different mode). To explicitly specify merge mode, the application can include the **<default-operation>** tag element with the value merge in the **<edit-config>** tag element:

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

The client application can specify one of two alternative default modes for incorporating new configuration data:

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

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

```
]]>]]>
```

We recommend using replace mode only when you completely overwrite the candidate configuration with new configuration data. When the default mode is replace, we do not recommend including the **operation** attribute on individual configuration elements in the new configuration to specify a different incorporation mode for them.

It is also possible to replace individual configuration elements while merging or creating others. See "Replacing Configuration Elements" on page 76.

- In *no-change mode*, configuration elements in the existing candidate configuration remain unchanged unless the new configuration includes a corresponding element that has the **operation** attribute in its opening container tag to specify an incorporation mode. This mode prevents the NETCONF server from creating parent hierarchy levels for an element that is being deleted. For more information, see "Deleting Configuration Elements" on page 78. To **specify no-change mode**, the candidate application includes the **<default-operation>** tag element with the value none in the **<edit-config>** tag element:

```
<rpc>
    <edit-config>
      <default-operation>none</default-operation>
        <!- - other child tag elements of the <edit-config> tag element -
->
    </edit-config>
</rpc>
]]>]]>
```

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

Replacing the Entire Candidate Configuration

A client application can completely replace the current candidate configuration, either with new data or by rolling back to a previous configuration.

NOTE: To comply with the NETCONF specification, the NETCONF server accepts the **<delete-config>** tag element, which deletes the entire candidate configuration. However, a commit operation fails if the candidate configuration does not exist or is completely empty, so the application must use the **<edit-config>** or **<copy-config>** tag element to add data to the candidate configuration before committing it. See "<delete-config>" on page 90.

For information about completely replacing the candidate configuration, see the following sections:

- Replacing the Candidate Configuration with Newly Defined Data on page 72
- Replacing the Candidate Configuration with the Running Configuration on page 73

Replacing the Candidate Configuration with Newly Defined Data

To replace the entire candidate configuration with new configuration data, a client application can use either of two methods, as described in the following sections.

With either method, the NETCONF server confirms that it replaced the candidate configuration by returning the **<ok/>** tag in the **<rpc-reply>** tag element:

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

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

Replacing the Configuration with the Contents of a File

One method for replacing the entire candidate configuration is to include the **<copy-config>** tag element in the **<rpc>** tag element. The **<source>** tag element encloses the **<url>** tag element to specify the filename that contains the new configuration data. The **<target>** tag element encloses the **<candidate/>** tag to indicate that the new configuration data replaces the candidate configuration:

```
<rpc>
      <copy-config>
        <target>
            <candidate/>
        </target>
        <source>
            <url>
                <!- - location specifier for file containing the new
configuration - ->
            </url>
        </source>
      </copy-config>
</rpc>
]]>]]>
```

Setting Replace Mode as the Default Mode

The other method for replacing the entire candidate configuration is to set replace mode as the default incorporation mode. The candidate configuration includes the **<default-operation>**tag element with the value replace in the **<edit-config>** tag element, as described in "Setting the Default Mode for Incorporating New Configuration Data" on page 70. To specify the new configuration data, the application includes either a **<config>**

tag element that contains the data or a **<url>** tag element that names the file containing the data, as discussed in "Defining the New Configuration Data" on page 67.

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

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

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

Replacing the Candidate Configuration with the Running Configuration

To discard changes made to the candidate configuration and make its contents match the contents of the current running (active) configuration, a client application includes the **<discard-changes/>** tag in an**<rpc>** tag element:

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

This operation is equivalent to the CLI configuration mode **rollback** command.

The NETCONF server indicates that it discarded the changes by enclosing the **<ok/>** tag in the **<rpc-reply>** tag element:

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

Changing Individual Configuration Elements

Although the NETCONF server by default merges new configuration data into the existing candidate configuration, a client application can also replace, create, or delete individual configuration elements (hierarchy levels or configuration objects). The same basic tag elements are emitted for all operations—the **<edit-config>**, **<target>**, and either **<config>** or **<url>** tag elements, plus the **<candidate/>** tag, in an **<rpc>** tag element:

```
<rpc>
    <edit-config>
```

```
        <target>
            <candidate/>
        </target>

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

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

Within the **<config>** tag element or in the file named by the **<url>** tag element, the application defines a configuration element by including 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 element. To represent the element, the application includes its container tag element. The child tag elements included within the container tag element depend on the operation, and are described in the following sections.

For more information about the tag elements that represent configuration statements, see "Mapping Configuration Statements to SRC XML Tag Elements" on page 15. For information about the tag elements for a specific configuration element, see the *SRC XML API Configuration Reference*.

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

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

For more information, see the following sections:

- Merging Configuration Elements on page 74
- Replacing Configuration Elements on page 76
- Creating New Configuration Elements on page 77
- Deleting Configuration Elements on page 78

Merging Configuration Elements

To merge configuration elements (hierarchy levels or configuration objects) into the candidate configuration, a client application emits the basic tag elements described in "Changing Individual Configuration Elements" on page 73.

To represent each element to merge (either within the **<config>** tag element or in the file named by the **<url>** tag element), the application includes the tag elements representing its parent hierarchy levels and its container tag element, as described in "Changing Individual Configuration Elements" on page 73. Within the container tag, the application includes each of the element's identifier tag elements (if it has them) and the tag element for each child to add or for which to set a different value. In the following, the identifier tag element is called **<name>**:

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

The NETCONF server merges the new configuration element according to the rules specified in "Setting the Default Mode for Incorporating New Configuration Data" on page 70. As described in that section, the application can explicitly specify merge mode by including the **<default-operation>**tag element with the value **merge** in the **<edit-config>** tag element.

The following example shows how to merge information for a new interface called **eth1** into the **[edit interfaces]** hierarchy level in the candidate configuration:

Client Application	NETCONF Server

```
<rpc>
 <edit-config>
  <target>
   <candidate/>
  </target>
  <config>
   <configuration>
    <interfaces>
     <interface>
      <name>eth1</name>
       <unit>
        <name>0</name>
        <family>
         <inet>
          <address>10.0.0.1/8</address>
         </inet>
        </family>
       </unit>
     </interface>
    </interfaces>
   </configuration>
  </config>
 </edit-config>
</rpc>
]]>]]>
```

Client Application	NETCONF Server
	<rpc-reply xmlns="*URN*" xmlns:sdx=" *URL*" > <ok/> </rpc-reply>]]>]]>

Replacing Configuration Elements

To replace configuration elements (hierarchy levels or configuration objects) in the candidate configuration, a client application emits the basic tag elements described in "Changing Individual Configuration Elements" on page 73.

To represent the new definition for each configuration element being replaced (either within the **<config>** tag element or in the file named by the **<url>** tag element), the application emits the tag elements representing its parent hierarchy levels and its container tag element, as described in "Changing Individual Configuration Elements" on page 73. Within the container tag, the application includes each of the element's identifier tag elements (if it has them) and all child tag elements (with values if appropriate) that are being defined for the new version of the element. In the following, the identifier tag element is called **<name>**. The application includes the **operation="replace"** attribute in the opening container tag:

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

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

The application can also replace all objects in the configuration in one operation. For instructions, see "Replacing the Entire Candidate Configuration" on page 71.

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

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <config> <configuration> <system> <login> <class operation=" replace" > <name>operator</name> <permissions>configure</permissions> <permissions>admin-control</permissions> </class> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Creating New Configuration Elements

To create configuration elements (hierarchy levels or configuration objects) in the candidate configuration only if the elements do not already exist, a client application emits the basic tag elements described in "Changing Individual Configuration Elements" on page 73.

To represent each configuration element being created (either within the **<config>**tag element or in the file named by the **<url>**tag element), the application emits the tag elements representing its parent hierarchy levels and its container tag element, as described in "Changing Individual Configuration Elements" on page 73. Within the container tag, the application includes each of the element's identifier tag elements (if it has them) and all child tag elements (with values if appropriate) that are being defined for the element. In the following, the identifier tag element is called **<name>**. The application includes the **operation="create"** attribute in the opening container tag:

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

```
            <!- - other child tag elements - ->
        </element>
    <!- - closing tags for each parent of the element - ->
</configuration>
```

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

The following example shows how to add a user to a C Series Controller if it is not already configured:

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <config> <configuration> <system> <login> <user operation=" create" > <name>camryn</name> <class>super-user</class> </user> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Deleting Configuration Elements

To delete a configuration element (hierarchy level or configuration object) from the candidate configuration, a client application emits the basic tag elements described in "Changing Individual Configuration Elements" on page 73. It also emits the **<default-operation>** tag element with the value none to change the default mode to no-change.

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

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

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

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

To represent each configuration element being deleted (either within the **<config>** tag element or in the file named by the **<url>** tag element), the application emits the tag elements representing its parent hierarchy levels, as described in "Changing Individual Configuration Elements" on page 73. The tag element in which the **operation="delete"** attribute is included depends on the element type, as described in the following sections:

- Deleting a Hierarchy Level or Container Object on page 79
- Deleting a Configuration Object That Has an Identifier on page 80
- Deleting a Single-Value or Fixed-Form Option from a Configuration Object on page 81
- Deleting Values from a Multivalue Option of a Configuration Object on page 82

Deleting a Hierarchy Level or Container Object

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

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

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value none, as described in "Deleting Configuration Elements" on page 78. For more information about hierarchy levels and container objects, see "Mapping for Hierarchy Levels and Container Statements" on page 15.

The following example shows how to remove the **[edit system services netconf]** hierarchy level of the candidate configuration:

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <default-operation>none</default-operation> <config> <configuration> <system> <services> <netconf operation=" delete" /> </services> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Deleting a Configuration Object That Has an Identifier

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

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

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

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value none, as described in "Deleting

Configuration Elements" on page 78. For more information about identifiers, see "Mapping for Objects That Have an Identifier" on page 15.

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

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <default-operation>none</default-operation> <config> <configuration> <system> <login> <user operation=" delete" > <name>barbara</name> </user> </login> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

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

To delete from a configuration object either a fixed-form option or an option that takes just one value, a client application includes the **operation="delete"** attribute in the tag element for the option. In the following, the identifier tag element for the object is called **<name>**. (For information about deleting an option that can take multiple values, see "Deleting Values from a Multivalue Option of a Configuration Object" on page 82.)

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

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value none, as described in "Deleting

Configuration Elements" on page 78. For more information about options, see "Mapping for Single-Value and Fixed-Form Leaf Statements" on page 17.

The following example shows how to remove the fixed-form **stand-alone** option at the **[edit system ldap server]** hierarchy level:

Client Application	NETCONF Server
<rpc> <edit-config> <target> <candidate/> </target> <default-operation>none</default-operation> <config> <configuration> <system> <ldap> <server> <stand-alone operation=" delete" /> </server> </ldap> </system> </configuration> </config> </edit-config> </rpc>]]>]]>	
	<rpc-reply xmlns="*URN*" xmlns:sdx="*URL*" > <ok/> </rpc-reply>]]>]]>

Deleting Values from a Multivalue Option of a Configuration Object

As described in "Mapping for Leaf Statements with Multiple Values" on page 18, some 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 XML representation does not use a parent tag for the object, but instead uses a separate instance of the object tag element for each value. In the following, the identifier tag element is called **<name>**:

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

```
</parent-object>
```

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

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

We recommend that the application set the default mode to no-change by including the **<default-operation>** tag element with the value none, as described in "Deleting Configuration Elements" on page 78. For more information about leaf statements with multiple values, see "Mapping for Leaf Statements with Multiple Values" on page 18.

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

Client Application	NETCONF Server

```
<rpc>
 <edit-config>
  <target>
   <candidate/>
  </target>
  <default-operation>none</default-operation>
  <config>
   <configuration>
    <system>
     <login>
      <class>
       <name>user-accounts</name>
       <permissions operation=" delete" >configure</permissions>
       <permissions operation=" delete" >control</permissions>
      </class>
     </login>
    </system>
   </configuration>
  </config>
 </edit-config>
</rpc>
]]>]]>
```

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

CHAPTER 6

Committing Configurations

This chapter explains how to commit a candidate configuration so that it becomes the active configuration on the C Series Controller. For more detailed information about commit operations, including a discussion of the interaction among different variants of the operation, see the *SRC PE CLI User Guide*.

This chapter includes the following topics:

Verifying a Configuration Before Committing It

During the process of committing the candidate configuration or a private copy, the NETCONF server confirms that it is syntactically correct. If the syntax check fails, the server does not commit the candidate.

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

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

If the candidate is not valid, the **<rpc-reply>** tag element instead encloses an **<rpc-error>** tag element explaining the reason for the failure.

Committing a Configuration

To commit the candidate configuration, a client application includes the **<commit/>** tag in an **<rpc>** tag element:

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

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

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

If the NETCONF server cannot commit the candidate, the **<rpc-reply>** tag element instead encloses an **<rpc-error>** tag element explaining the reason for the failure. The most common causes are semantic or syntactic errors in the candidate configuration.

To avoid inadvertently committing changes made by other users or applications, a client application locks the candidate configuration before changing it and emits the **<commit/>** tag while the configuration is still locked. (For instructions on locking and changing the candidate configuration, see "Locking the Candidate Configuration" on page 38 and "Changing Configuration Information" on page 65.) After committing the configuration, the application unlocks the candidate as described in "Unlocking the Candidate Configuration" on page 39.

CHAPTER 7

Summary of NETCONF Tag Elements

This chapter lists the tag elements that client applications and the NETCONF server use to control the NETCONF session and to exchange configuration information. It also describes the **]]>]]>** character sequence, which signals the end of each request and response. The entries are in alphabetical order. For information about the notational conventions used in this chapter, see Table 2 on page xii.

]]>]]>

Usage

```
<hello>
    <!- - child tag elements included by client application or NETCONF server
 - ->
</hello>
]]>]]>

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

<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <!- - tag elements in the response from the NETCONF server - ->
</rpc-reply>
]]>]]>
```

Description

Signal the end of each XML document sent by the NETCONF server and client applications. Clients send the sequence after each XML document (after the closing **</hello>** tag and each closing **</rpc>** tag). The NETCONF server sends the sequence after its closing **</hello>** tag and each closing **</rpc-reply>** tag.

Use of this signal is required by RFC 4742, *Using the NETCONF Configuration Protocol over Secure Shell (SSH)*, available at http://www.ietf.org/rfc/rfc4742.txt.

Usage Guidelines

See "Generating Well-Formed XML Documents" on page 22.

Related Documentation

- <hello> on page 94
- <rpc> on page 97

- <rpc-reply> on page 98

<close-session/>

Usage

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

Description Request that the NETCONF server end the current session.

Usage Guidelines See "Ending a NETCONF Session and Closing the Connection" on page 41.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97

<commit>

Usage

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

Description Request that the NETCONF server commit the configuration immediately, making it the active configuration.

Usage Guidelines See "Committing a Configuration" on page 85.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97

<copy-config>

Usage

```
<rpc>
      <copy-config>
        <target>
            <candidate/>
        </target>
        <source>
            <url>
                <!- - location specifier for file containing the new
configuration - ->
            </url>
```

```
        </source>
      <copy-config>
</rpc>
]]>]]>
```

Description Replace the existing candidate configuration with configuration data contained in a file.

Contents <source>—Encloses the **<url>** tag element, which specifies the source of the configuration data.

<url>—Names the file that contains the new configuration data to substitute for the existing candidate configuration. For information about specifying the file location, see "Providing Configuration Data in a File" on page 67.

The **<target>** tag element and its contents are explained separately.

Usage Guidelines See "Replacing the Configuration with the Contents of a File" on page 72.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97
- <target> on page 99

<data>

Usage

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
<data>
        <configuration>
            <!- - XML tag elements for the configuration data - ->
        </configuration>
</data>
</rpc-reply>
]]>]]>
```

Description Enclose configuration data returned by the NETCONF server in response to a **<get-config>** tag element.

Contents <configuration>—Encloses configuration tag elements. It is the top-level tag element in the XML API, equivalent to the **[edit]** hierarchy level in the CLI. For information about configuration elements, see the *SRC XML API Configuration Reference*.

Usage Guidelines See "Requesting Configuration Information" on page 49.

Related Documentation

-]]>]]> on page 87

- <get-config> on page 93
- <rpc-reply> on page 98

<delete-config>

Usage

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

Description Delete the existing candidate configuration.

Contents The **<target>** tag element and its contents are explained separately.

Usage Guidelines See "Replacing the Entire Candidate Configuration" on page 71.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97
- <target> on page 99

<discard-changes/>

Usage

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

Description Discard changes made to the candidate configuration, and make its contents match the contents of the current running (active) configuration. This operation is equivalent to the CLI configuration mode **rollback** command.

Usage Guidelines See "Replacing the Candidate Configuration with the Running Configuration" on page 73.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97

<edit-config>

Usage

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

    <!- - EITHER - ->
        <config>
            <configuration>
                <!- - tag elements representing the data to incorporate - ->

            </configuration>
        </config>

    <!- - OR - ->
        <url>
            <!- - location specifier for file containing data - ->
        </url>

        <default-operation>(merge | none | replace)</default-operation>
<edit-config>
</rpc>
]]>]]>
```

Description Request that the NETCONF server incorporate configuration data into the candidate configuration. Provide the data in one of two ways:

- Include the **<url>** tag element to specify the location of a file that contains the XML configuration tag elements to incorporate.
- Include the **<config>** tag element to provide a data stream of XML configuration tag elements to incorporate. The tag elements are enclosed in the **<configuration>** tag element.

Contents <config>—Encloses the **<configuration>** tag element.

<configuration>—Encloses the tag elements to incorporate into the candidate configuration, provided as a data stream. For information about the syntax for representing the elements to create, delete, or modify, see "Mapping Configuration Statements to SRC XML Tag Elements" on page 15 and "Changing Individual Configuration Elements" on page 73.

<default-operation>—(Optional) Specifies how to incorporate the new configuration data into the candidate configuration, particularly when there are conflicting statements. The following are acceptable values:

- merge—Combines the new configuration data with the candidate configuration according to the rules defined in "Setting the Default Mode for Incorporating New Configuration Data" on page 70. This is the default mode if the **<default-operation>** tag element is omitted. It applies to all elements in the new data that do not have the **operation** attribute in their opening container tag to specify a different mode. (For information about the **operation** attribute, see "Changing Individual Configuration Elements" on page 73.)
- none—Retains each configuration element in the existing candidate configuration unless the new data includes a corresponding element that has the **operation** attribute in its opening container tag to specify an incorporation mode. This mode prevents the NETCONF server from creating parent hierarchy levels for an element that is being deleted. For more information, see "Deleting Configuration Elements" on page 78.
- replace—Discards the existing candidate configuration and replaces it with the new data. For more information, see "Setting Replace Mode as the Default Mode" on page 72.

<url>—Specifies the full pathname of the file that contains the configuration data to load. The file must reside on the local disk. For more information, see "Providing Configuration Data in a File" on page 67.

The **<target>** tag element and its contents are explained separately.

Usage Guidelines See "Changing Configuration Information" on page 65.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97
- <target> on page 99

<error-info>

Usage

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <rpc-error>
      <error-info>
        <bad-element>command-or-statement</bad-element>
      </error-info>
    </rpc-error>
</rpc-reply>
]]>]]>
```

Description Provide additional information about the event or condition that causes the NETCONF server to report an error or warning in the **<rpc-error>** tag element.

Contents <bad-element>—Identifies the command or configuration statement that was being processed when the error or warning occurred. For a configuration statement, the

<error-path> tag element enclosed in the **<rpc-error>** tag element specifies the statement's parent hierarchy level.

Usage Guidelines See "Handling an Error or Warning" on page 37.

Related Documentation

-]]>]]> on page 87
- <rpc-error> on page 97
- <rpc-reply> on page 98

<get-config>

Usage

```
<rpc>
<get-config>
        <source>
            <candidate/>
        </source>
</get-config>

<get-config>
        <source>
            <candidate/>
        </source>
        <filter type="subtree">
            <configuration>
                <!- - tag elements for each configuration element to return
- ->
            </configuration>
        </filter>
</get-config>
</rpc>
]]>]]>
```

Description Request configuration data from the NETCONF server. The child tag elements **<source>** and **<filter>** specify the source and scope of data to display:

- To display the entire candidate configuration, enclose the **<source>** tag element and **<candidate/>** tag in the **<get-config>** tag element.
- To display one or more sections of the configuration hierarchy (hierarchy levels or configuration objects), enclose the appropriate child tag elements in the **<source>** and **<filter>** tag elements.

Contents <candidate/>—Represents the candidate configuration.

<configuration>—Encloses tag elements that specify which configuration elements to return.

<filter>—Encloses the <configuration> tag element. The mandatory type attribute indicates the kind of syntax used to represent the requested configuration elements; the only acceptable value is **subtree**.

To specify the configuration elements to return, include within the **<filter>** tag element the XML tag elements that represent all levels of the configuration hierarchy from the root (represented by the **<configuration>** tag element) down to each element to display. For information about the syntax for representing each kind of element, see "Specifying the Scope of Configuration Information to Return" on page 51. For information about the configuration elements available in the current version of the SRC software, see the *SRC XML API Configuration Reference*.

<source>—Encloses the tag that specifies the source of the configuration data. To specify the candidate configuration, include the **<candidate/>**tag.

Usage Guidelines See "Requesting Configuration Information" on page 49.

Related Documentation

-]]>]]> on page 87
- <data> on page 89
- <rpc> on page 97

<hello>

Usage

```
<!- - emitted by a client application - ->
<hello>
    <capabilities>
        <capability>URI</capability>
    </capabilities>
</hello>
]]>]]>

<!- - emitted by the NETCONF server - ->
<hello>
    <capabilities>
        <capability>URI</capability>
    </capabilities>
    <session-id>session-identifier</session-id>
</hello>
]]>]]>
```

Description Specify which operations, or *capabilities*, the emitter supports from among those defined in the NETCONF specification. The client application must emit the **<hello>** tag element before any other tag element during the NETCONF session, and must not emit it more than once.

Contents <capabilities>—Encloses one or more **<capability>** tags, which together specify the set of supported NETCONF operations.

<capability>—Specifies the uniform resource identifier (URI) of a capability defined in the NETCONF specification or by a vendor. Each capability from the NETCONF specification is represented by a uniform resource name (URN). Capabilities defined by vendors are represented by URNs or URLs. For a list of the capabilities supported by the NETCONF server for the SRC software, see "Exchanging <hello> Tag Elements" on page 28.

<session-id>—(Generated by NETCONF server only) Specifies the process ID (PID) of the NETCONF server for the session.

Usage Guidelines See "Exchanging <hello> Tag Elements" on page 28.

Related Documentation

-]]>]]> on page 87

<kill-session>

Usage

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

Description Request that the NETCONF server terminate another NETCONF session. The usual reason to emit this tag is that the user or application for the other session holds a lock on the candidate configuration, preventing the client application from locking the configuration itself.

The client application must have **maintenance** permission.

Contents <session-id>—The PID of the entity conducting the session to terminate. The PID is reported in the **<rpc-error>** tag element that the NETCONF server generates when it cannot lock a configuration as requested.

Usage Guidelines See "Terminating Another NETCONF Session" on page 40.

Related Documentation

-]]>]]> on page 87
- <lock> on page 95
- <rpc> on page 97

<lock>

Usage

```
<rpc>
```



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

Description Request that the NETCONF server lock the candidate configuration, enabling the client application both to read and change it, but preventing any other users or applications from changing it. The application must emit the **<unlock/>** tag to unlock the configuration.

If the NETCONF session ends or the application emits the **<unlock>** tag element before the candidate configuration is committed, all changes made to the candidate are discarded.

Contents The **<target>** tag element and its contents are explained separately.

Usage Guidelines See "Locking the Candidate Configuration" on page 38.

Related Documentation

-]]>]]> on page 87
- <rpc> on page 97
- <target> on page 99
- <unlock> on page 99

<ok/>

Usage

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

Description Indicate that the NETCONF server successfully performed a requested operation that changes the state or contents of the configuration.

Usage Guidelines See "Configuration Change Responses" on page 36.

Related Documentation

-]]>]]> on page 87
- <rpc-reply> on page 98

<rpc>

Usage

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

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"***. This feature can be used to associate requests and responses if the value assigned to an attribute by the client application is unique in each opening **<rpc>** tag. The NETCONF server echoes the attribute unchanged in its opening **<rpc-reply>** tag, making it simple to map the response to the initiating request. The NETCONF specification assigns the name **message-id** to this attribute.

Usage Guidelines See "Sending a Request to the NETCONF Server" on page 31.

Related Documentation

-]]>]]> on page 87
- <rpc-reply> on page 98

<rpc-error>

Usage

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
      <rpc-error>
        <error-severity>error-severity</error-severity>
        <error-path>error-path</error-path>
        <error-message>error-message</error-message>
        <error-info>...</error-info>
      </rpc-error>
</rpc-reply>
]]>]]>
```

Description Indicate that the NETCONF server has experienced an error while processing the client application's request. If the server has already emitted the response tag element for the current request, the information enclosed in that response tag element might be incomplete. The client application must include code that discards or retains the information, as appropriate. The child tag elements described in the Contents section detail the nature of the error. The NETCONF server does not necessarily emit all child tag elements; it omits tag elements that are not relevant to the current request.

Contents <error-message>—Describes the error or warning in a natural-language text string.

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

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

The **<error-info>** tag element is described separately.

Usage Guidelines See "Handling an Error or Warning" on page 37.

Related Documentation

-]]>]]> on page 87
- <error-info> on page 92
- <rpc-reply> on page 98

<rpc-reply>

Usage

```
<rpc-reply xmlns="URN" xmlns:sdx="URL">
    <!- - tag elements in a reply from the NETCONF server - ->
</rpc-reply>
]]>]]>
```

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

- The XML tag element that encloses the data requested by a client application with an XML operational request tag element; for example, the **<interface-information>** tag element in response to the **<get-interface-information>** tag element
- The **<data>** tag element, to enclose the data requested by a client application with the **<get-config>** tag element
- The **<ok/>** tag, to confirm that the NETCONF server successfully performed an operation that changes the state or contents of a configuration (such as a lock, change, or commit operation)
- The **<output>** tag element, if the XML API does not define a specific tag element for requested operational information
- The **<rpc-error>** tag element, if the requested operation generated an error or warning

Attributes xmlns—Names the default XML namespace for the enclosed tag elements.

Usage Guidelines See "Parsing the NETCONF Server Response" on page 34.

Related Documentation

-]]>]]> on page 87
- <data> on page 89
- <ok/> on page 96

- <rpc> on page 97
- <rpc-error> on page 97

<target>

Usage

```
<rpc>
    <( copy-config | delete-config | edit-config | lock | unlock )>
        <target>
            <candidate/>
        </target>
    </( copy-config | delete-config | edit-config | lock | unlock )>
</rpc>
]]>]]>
```

Description Specify the configuration on which to perform an operation.

Contents <candidate/>—Specifies the candidate configuration as the configuration on which to perform the operation. This is the only acceptable value for the SRC software.

Usage Guidelines See "Locking the Candidate Configuration" on page 38, "Unlocking the Candidate Configuration" on page 39, "Changing the Candidate Configuration" on page 66, and "Replacing the Configuration with the Contents of a File" on page 72.

Related Documentation

-]]>]]> on page 87
- <copy-config> on page 88
- <delete-config> on page 90
- <edit-config> on page 91
- <lock> on page 95
- <rpc> on page 97
- <unlock> on page 99

<unlock>

Usage

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

Description Request that the NETCONF server unlock and close the candidate configuration, which the client application previously locked by emitting the **<lock>** tag. Until the application emits this tag element, other users or applications can read the configuration but cannot change it.

Contents The **<target>** tag element and its contents are explained separately.

Usage Guidelines See "Unlocking the Candidate Configuration" on page 39.

Related Documentation

-]]>]]> on page 87
- <lock> on page 95
- <rpc> on page 97
- <target> on page 99

CHAPTER 8

Summary of Attributes in SRC XML Tags

This chapter describes the attributes that the NETCONF server and client applications include in opening SRC XML tags. For information about the notational conventions used in this chapter, see Table 2 on page xii.

operation

Usage

```
<rpc>
    <edit-config>
        <config>
            <configuration>
                <!- - opening tags for each parent of the changing element -
 ->
                    <changing-element operation="( create | delete | replace
 ) ">
                        <name>identifier</name> <!- - if changing element has
 an identifier - ->
                         <!- - other child tag elements, if appropriate for
the operation - ->
                    </changing-element>
                <!- - closing tags for each parent of the changing element -
 ->
            </configuration>
        </config>
        <!- - other child tag elements of the <edit-config> tag element - ->
    <edit-config>
</rpc>
]]>]]>
```

Description Specify how the NETCONF server incorporates an individual configuration element into the candidate configuration. If the attribute is omitted, the element is merged into the configuration according to the rules defined in "Setting the Default Mode for Incorporating New Configuration Data" on page 70. The following are acceptable values:

- create—Creates the specified element in the configuration only if the element does not already exist. See "Creating New Configuration Elements" on page 77.
- delete—Deletes the specified element from the candidate configuration. We recommend that the **<default-operation>** tag element with the value **none** also be included in the **<edit-config>** tag element. See "Deleting Configuration Elements" on page 78.

- replace—Replaces the specified element in the candidate configuration with the provided new configuration data. See "Replacing Configuration Elements" on page 76.

Usage Guidelines See "Changing Individual Configuration Elements" on page 73.

Related Documentation

- <edit-config> on page 91
- <rpc> on page 97
- xmlns on page 103

sdx:changed-localtime

Usage

```
<rpc-reply xmlns:sdx="URL">
    <configuration xmlns="URL" sdx:changed-seconds=" seconds" \
                sdx:changed-localtime="YYYY-MM-DD hh:mm:ss TZ">
        <!- - 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 C Series Controller's local time zone.

Usage Guidelines See "Requesting Information from the Candidate Configuration" on page 51.

Related Documentation

- <rpc-reply> on page 98
- sdx:changed-seconds on page 102
- xmlns on page 103

sdx:changed-seconds

Usage

```
<rpc-reply xmlns:sdx="URL">
    <configuration xmlns="URL" sdx:changed-seconds="seconds" \
                sdx:changed-localtime="YYYY-MM-DD hh:mm:ss TZ">
        <!- - 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.

Usage Guidelines See "Requesting Information from the Candidate Configuration" on page 51.

Related Documentation

- <rpc-reply> on page 98
- operation on page 101
- xmlns on page 103

xmlns

Usage

```
<rpc-reply xmlns:sdx="URL">
   <operational-response xmlns="URL-for-DTD">
        <!- -XML tag elements for the requested operational data - ->
    </operational-response>
</rpc-reply>

<rpc-reply xmlns:sdx="URL">
    <configuration xmlns="URL" sdx:changed-seconds="seconds" \
                sdx:changed-localtime="YYYY-MM-DD hh:mm:ss TZ" >
        <!- - XML tag elements for the requested configuration data - ->
    </configuration>
</rpc-reply>
```

Description

For operational responses, define the XML namespace for the enclosed tag elements that do not have a prefix (such as **sdx:**) in their names. The namespace indicates which XML document type definition (DTD) defines the set of tag elements in the response.

For configuration data responses, define the XML namespace for the enclosed tag elements.

Usage Guidelines

See "Requesting Operational Information" on page 48 and "Requesting Information from the Candidate Configuration" on page 51.

Related Documentation

- <rpc-reply> on page 98
- operation on page 101
- sdx:changed-seconds on page 102