

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 on page 2
- Advantages of Using the NETCONF and SRC XML APIs on page 3
- Overview of a NETCONF Session on page 5

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:

- XML and NETCONF Tag Elements on page 2
- Document Type Definition on page 3

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.

Overview of a NETCONF Session

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 19.

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.

