Junos® OS

NETCONF PHP TOOLKIT GUIDE

Abbreviated Table of Contents

About This Guide

Part 1 Overview and Installation

Chapter 1 NETCONF PHP Toolkit Overview 5

Chapter 2 NETCONF PHP Toolkit Installation 8

Part 2 Using the NETCONF PHP Toolkit

Chapter 3 NETCONF PHP Toolkit Classes 13

Chapter 4 NETCONF PHP Toolkit Program Files 21

Chapter 5 Performing Operational and Configuration Tasks 32

Chapter 6 NETCONF PHP Toolkit Examples 46

FAQs.....51

List of Tables

Part 1 Overview and Installation

Chapter 1 NETCONF PHP Toolkit Overview 4
Table 1: NETCONF PHP Toolkit Classes 5

Part 2 Using the NETCONF PHP Toolkit

Chapter 3 NETCONF PHP Toolkit Classes 12
Table 2: Creating a Configuration Hierarchy with XMLBuilder and XML 16

PART 1

Overview and Installation

- NETCONF PHP Toolkit Overview
- NETCONF PHP Toolkit Installation

CHAPTER 1

NETCONF PHP TOOLKIT OVERVIEW

- NETCONF XML Management Protocol and Junos XML API Overview
- NETCONF PHP Toolkit Overview

NETCONF XML Management Protocol and Junos XML API Overview

The NETCONF XML management protocol is an XML-based protocol that client applications use to request and change configuration information on routing, switching, and security devices. The NETCONF XML management protocol uses an Extensible Markup Language (XML)-based data encoding for the configuration data and remote procedure calls. The NETCONF protocol defines basic operations that are equivalent to configuration mode commands in the command-line interface (CLI). Applications use the protocol operations to display, edit, and commit configuration statements (among other operations), just as administrators use CLI configuration mode commands to perform those operations. The Junos XML API is an XML representation of Junos OS configuration statements and operational mode commands. When the client application manages a device running Junos OS, Junos XML configuration tag elements are the content to which the NETCONF XML protocol operations apply. Junos XML operational tag elements are equivalent in function to operational mode commands in the CLI, which administrators use to retrieve status information for a routing platform running Junos OS.

The NETCONF XML management protocol is described in RFC 4741, *NETCONF Configuration Protocol*, which is available at http://www.ietf.org/rfc/rfc4741.txt .Client applications request or change information on a switch, router, or security device by encoding the request with tag elements from the NETCONF XML management protocol and Junos XML API and then sending it to the NETCONF server on the device. On devices running Junos OS, the NETCONF server is integrated into Junos OS and does not appear as a separate entry in process listings. The NETCONF server directs the request to the appropriate software modules within the device, encodes the response in NETCONF and Junos XML API tag elements, and returns the result to the client application. For example, to request information about the status of a device's interfaces, a client application sends the <get-interface-information> tag element from the Junos XML API. The NETCONF server gathers the information from the interface process and returns it in the
<interface-information> tag element.

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

NETCONF PHP Toolkit Overview

The NETCONF PHP toolkit provides an object-oriented interface for communicating with a NETCONF server. The toolkit enables programmers familiar with the PHP programming language to easily connect to a device, open a NETCONF session, construct configuration

hierarchies in XML, and create and execute operational and configuration requests. You can create your own PHP applications to manage and configure routing, switching, and security devices.

The NETCONF PHP toolkit provides classes with methods that implement the functionality of the NETCONF protocol operations defined in RFC 4741. All basic protocol operations are supported. The NETCONF XML management protocol uses XML-based data encoding for configuration data and remote procedure calls. The toolkit provides classes and methods that aid in creating, modifying, and parsing XML.
The NETCONF PHP toolkit has three basic classes which are described in table no 1.

Table No. 1 NETCONF PHP Toolkit Classes

CLASS	SUMMARY
Device	Defines the device on which NETCONF server runs, represents SSHv2 connection and NETCONF session with that device
XMLBuilder	It creates an XML object and define functions to create new configuration, new RPC and new XML as an XML object.
XML	Represents XML contents and provide methods to manipulate it.

A *configuration management server* is generally a PC or workstation that is used to configure a router, switch, or security device remotely. The communication between the configuration management server and the NETCONF server via the NETCONF PHP toolkit involves:

1. Establishing a NETCONF session over SSHv2 between the configuration management server and the NETCONF server.
2. Creating RPCs corresponding to requests and sending these requests to the NETCONF server.
3. Receiving and processing the RPC replies from the NETCONF server.

To use the NETCONF PHP toolkit, you must install the toolkit by following the steps mentioned in "Downloading and Installing the NETCONF PHP Toolkit"

Once the toolkit is installed, you can connect to a device, create a NETCONF session, and execute operations by adding the associated code to a php program file, which is then executed.
For more information about creating NETCONF PHP toolkit programs, see "Creating and Executing a NETCONF PHP Toolkit Program File"

Related Documentation

- NETCONF PHP Toolkit Class: Device
- NETCONF PHP Toolkit Class: XML
- NETCONF PHP Toolkit Class: XMLBuilder
- NETCONF XML Management Protocol and Junos XML API Overview
- Downloading and Installing the NETCONF PHP Toolkit
- Creating and Executing a NETCONF PHP Toolkit Program File

Chapter 2

NETCONF PHP Toolkit Installation

- Downloading and Installing the NETCONF PHP Toolkit

Downloading and Installing the NETCONF PHP Toolkit

The NETCONF PHP APIs depends on expect package of php http://pecl.php.net/package/expect

Setting up system (installing PHP and its extension) before installing NETCONF PHP APIs in Ubuntu (all steps are mandatory)

1. Install PHP
   `apt-get install php5` (current stable php version).

2. Install PHP Development Framework
   `apt-get install php5-dev` (current stable php version).

   to check whether pear is installed properly or not, type command pear, there you will get list of command like discover-channel, upgrade,upgrade-all, if pear is successfully installed.

3. Install php-pear (PHP Extension and Application Repository), required install php extensions
   `apt-get install php-pear`

4. Install Tool command language (TCL) and TK, required for installing expect package
   - `apt-get install tcl tcl-dev`
   - `apt-get install tk tk-dev`

5. Install Expect package which will be needed in building php-expect
   `apt-get install expect expect-dev`

6. Install php-expect using pecl, PECL is a repository for PHP Extensions, providing a directory of all known extensions and hosting facilities for downloading and development of PHP extensions. The packaging and distribution system used by PECL is shared with, PEAR
   `pecl install expect`

   Sometime it may give error regarding channel, then use the path mentioned in the suggestion like
   `pecl install channel://pecl.php.net/expect-0.3.1`

   After successful installation of expect package, add the following line under extension in php.ini file (this will also be shown after successful installation of expect function)
   `extension = expect.so`

Note: These installation steps are tested in Ubuntu version 12.04LTS and greater.

Setting up system in Fedora/Centos before installing NETCONF PHP APIs

1. Install PHP and its Development Framework
   ```
   yum install php php-devel php-common
   ```

2. Install php-pear (PHP Extension and Application Repository),required to install php extensions
   ```
   yum install php-pear
   ```

   to check whether pear is installed properly or not, type command pear, there you will get list of command like discover-channel, upgrade,upgrade-all, if pear is successfully installed.

3. Install Tool command language (TCL) and TK, it is necessary for installing expect package
   - `yum install tcl tcl-devel`
   - `yum install tk tk-devel`

4. Install Expect package which will be needed in building php-expect
   ```
   yum install expect expect-devel
   ```

5. Install php-expect using pecl, *PECL is a repository for PHP Extensions, providing a directory of all known extensions and hosting facilities for downloading and development of PHP extensions. The packaging and distribution system used by* PECL is shared with, PEAR
   ```
   pecl install expect
   ```

   sometime it may give error regarding channel, then use the path mentioned in the suggestion like
   ```
   pecl install channel://pecl.php.net/expect-0.3.1
   ```

   After successful installation of expect package, add the following line under extension in php.ini file (this will also be shown after successful installation of expect function)
   ```
   extension = expect.so
   ```

Note: These installation steps are successfully tested in Fedora 15 i686 and Centos-6.5-i386.

Now your system is ready to download and install NETCONF PHP APIs

Steps for installing NETCONF PHP APIs

Download the source code from github at any path in your Desktop and then copy netconf folder in default php path i.e. /usr/share/php

1. `wget -O /any/path/netconf-php-master.zip https://github.com/Juniper/netconf-php/archive/master.zip`

   (usually default path of php is /usr/share/php)

```
2. unzip netconf-php-master.zip
3. copy netconf folder in default php path i.e. /usr/share
```

include this path in your program for Device.php

For example while writing your code, include path

```
include("netconf/Device.php")
```

Satisfying Requirements for SSHv2 Connections

The NETCONF server communicates with client applications within the context of a NETCONF session. The server and client explicitly establish a connection and session before exchanging data, and close the session and connection when they are finished.
The NETCONF PHP toolkit accesses the NETCONF server using the SSH protocol and uses the standard SSH authentication mechanism. To establish an SSHv2 connection with a device running Junos OS, you must ensure that the following requirements are met:

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

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

Related Documentation

- Creating and Executing a NETCONF PHP Toolkit Program File
- NETCONF PHP Toolkit Overview
- NETCONF XML Management Protocol and Junos XML API Overview

PART 2

Using the NETCONF PHP Toolkit

- NETCONF PHP Toolkit classes
- NETCONF PHP Toolkit program files
- Performing Operational and Configuration Task
- NETCONF PHP Toolkit Examples

Chapter 3

NETCONF PHP Toolkit Classes

- NETCONF PHP Toolkit class: Device
- NETCONF PHP Toolkit Class: XML
- NETCONF PHP Toolkit Class: XMLBuilder

NETCONF PHP Toolkit Class: Device

A **Device** object represents an SSHv2 connection and a default NETCONF session between the configuration management server and the device on which the NETCONF server resides.

When creating a Device object, you must provide the IP address or DNS name and the authentication details to create the SSHv2 connection. Authentication can be **user-password** based or **RSA/DSA key-based (public-private)**. You also have the option of specifying the port number for the SSHv2 connection and the client capabilities to be sent to the NETCONF server.

The constructor syntax, for user-password based authentication:

```
Device ($hostname, $username, $password)
Device($hostname, $username, $password, $port)
Device($hostname, $username, $password, $capabilities)
Device($hostname, $username, $password, $port, $capabilities)
```

The constructor parameters are:

- `hostname` – *(required)* IP address or DNS name of the device on which the NETCONF server is running and to which connect via SSHv2.
- `Username` – *(required)* Username for the login account on the device on which the NETCONF server is running.
- `Password` - *(required)* password for user-password based authentication or key-based authentication. If no password is required for key-based authentication, leave this argument.
- `Port` - *(optional)* Port number on which to establish the SSHv2 connection. The default port is 830. If you are connecting to a device that is configured for NETCONF over SSH on a port other than the default port, you must specify that port number in the arguments.
- `Capabilities` - *(optional)* Client capabilities to be communicated to the NETCONF server, if the capabilities are other than the default capabilities.

The default NETCONF capabilities sent to server are:
`"urn: ietf:params:xml:ns:netconf:base:1.0";`

```
"urn:ietf:params:xml:ns:netconf:base:1.0#candidate";
"urn:ietf:params:xml:ns:netconf:base:1.0#confirmed-
                                                commit";
"urn:ietf:params:xml:ns:netconf:base:1.0#validate";
"urn:ietf:params:xml:ns:netconf:base:1.0#url?protocol=
 http, ftp,file";
```

The general syntax for creating a Device object is:

```
$device_name = new Device ($hostname, $username,
                  $password, <port>, <capabilities>);
```

By default, a NetConfSession object is created when you create a new instance of Device and connect to a NETCONF server. Once you have created a Device object, you can perform NETCONF operations.

Example: The following example creates a Device object with an authenticated SSHv2 connection to IP address 10.10.1.1. The connection uses user password-based authentication with the login name "admin" and the password "PaSsWoRd". When the connect() method is called, it connects to the device and automatically establishes a default NETCONF session.

```
$my_device = new Device("10.10.1.1", "admin", "PaSsWoRd", null);
$my_device->connect();
```

To create a Device object with a NETCONF-over-SSH connection on port 49000 instead of the default port 830, add the port number to the constructor arguments.

```
$my_device = new Device("10.10.1.1", "admin", "PaSsWoRd", 49000);
```

Related Documentation

- Creating and Executing a NETCONF PHP Toolkit Program File
- Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program
- NETCONF PHP Toolkit Class: XML
- NETCONF PHP Toolkit Class: XMLBuilder

NETCONF PHP Toolkit Class: XMLBuilder

In a NETCONF session, communication between the configuration management server and the NETCONF server is through XML-encoded data. The configuration management server sends remote procedure calls (RPCs) to the NETCONF server, and the NETCONF server processes the RPC and returns an RPC reply. The **XMLBuilder (netconf/XMLBuilder.php)** and **XML(netconf/XML.php)** objects help create and parse XML-encoded data.
You use the XMLBuilder object to create a new XML object. The constructor syntax is:

```
$builder= new XMLBuilder ().
```

The XMLBuilder class includes methods to create a configuration hierarchy, an RPC, or an XML object as XML-encoded data. Each method is overloaded to accept multiple hierarchy levels. The methods return an XML object. For example, the methods to construct a configuration, RPC, or XML object with a single-tier hierarchy are:

- create_new_config(String *elementLevelOne*)
- create_new_rpc(String *elementLevelOne*)

- Create_new_xml(String *elementLevelOne*)

The following sample code creates a new XMLBuilder object, $builder. The XMLBuilder object calls the create_new_config() method to construct a three-tier configuration hierarchy consisting of a "security" element, a "policies" element child tag, and a "policy" element that is a child of "policies".

```
XMLBuilder $builder = new XMLBuilder();
XML policy = $builder->create_new_config ("security", "policies",
"policy");
```

The resulting XML hierarchy is as follows:

```
<configuration>
    <security>
          <policies>
                <policy>

                </policy>
          </policies>
    </security>
</configuration>
```

Notice that the `create_new_config()` method always encloses the hierarchy within a top-level root element `<configuration>`. Similarly, the `create_new_rpc()` method encloses the hierarchy within an `<rpc>` tag element.
Once you generate an XML object, you can call methods from the XML class to manipulate that object.

Related Documentation:

- NETCONF PHP Toolkit class: Device
- NETCONF PHP Toolkit class: XML
- NETCONF PHP Toolkit Overview

NETCONF PHP Toolkit Class: XML

A **XML** object represents **XML-encoded data** and provides methods to modify and parse the XML. An XML object represents XML content and provides methods to manipulate it.The XML object has an 'active' element, which represents the hierarchy at which the XML can be manipulated.

For Example, if you want to load XML configuration on a Device then

- Create an `XMLBuilderObject`.
- Create a configuration as an XML object.
- Call the `execute_rpc(XML)` method on Device

It is recommended that you work with the XML object to create new configurations, remote procedure calls (RPCs), or any XML-based data. Using an XML object, you can easily add, delete, or modify elements and attributes. To facilitate modification of XML content, the XML

object maintains an 'active' element, which represents the hierarchy level exposed for modification.

To create an XML object, you first create an XMLBuilder object and construct the initial XML hierarchy. The XMLBuilder methods return an XML object on which you can then build. This makes it convenient to create XML-based configurations and RPCs and also parse the XML-based replies received from the NETCONF server.

Example: This example creates the following sample XML configuration hierarchy. The steps used
Configuration to create the configuration hierarchy are outlined in Table No 2
Hierarchy

```
<configuration>
      <security>
            <policies>
                  <policy>
                        <from-zone-name>trust</from-
                        zone-name>
                        <to-zone-name>untrust</to-zone-
                        name>
                  <policy>
                       <name>my-sec-policy</name>
                  <match>
            <source-address>any</source-address>
                        <destination-
                        address>any</destinationaddress>
                  <application>junos-ftp</application>
                  <application>junos-ntp</application>
                  <application>junos-ssh</application>
                  </match>
                  <then>
                  <permit>
                  </permit>
                  </then>
                  </policy>
                  </policy>
            </policies>
      </security>
</configuration>
```

Table No 2 Creating a configuration Hierarchy with XMLBuilder and XML objects

PHP CODE	RESULTING HIERARCHY

`$builder = new XMLBuilder();`
`$policy =builder->create_new_config`
`("security","policies","policy");`
	<configuration>
	<security>
	<policies>
	<policy>
	</policy>
	</policies>
	</security>
	</configuration>

`// Append nodes at the 'policy' level`
 	
`$policy->append("from-zone-name","trust");`
`$policy->append("to-zone-name","untrust");`
	<configuration>
	<security>
	<policies>
	<policy>
	<from-zone-name>trust</from-zone-name>
	<to-zone-name>untrust</to-zone-name>
	</policy>
	</policies>
	</security>
	</configuration>

`// Create a new hierarchy level for the first policy`
 	
`$policyOne = $policy->append("policy");`
`$policyOne->append("name","my-sec-policy");`
	<configuration>
	<security>
	<policies>
	<policy>
	<from-zone-name>trust</from-zone-name>
	<to-zone-name>untrust</to-zone-name>
	<policy>
	<name>my-sec-policy</name>
	</policy>
	</policy>
	</policies>
	</security>
	</configuration>

`// Create the 'match' hierarchy`
`$match = $policyOne->append("match");`
 	
`// Create and append an applications array`
 	
`$match->append("application", "junos-ftp,junos-ntp,junos-ssh");`
	<configuration>
	<security>
	<policies>
	<policy>
	<from-zone-name>trust</from-zone-name>
	<to-zone-name>untrust</to-zone-name>
	<policy>
	<name>my-sec-policy</name>
	<match>
	<application>junos-ftp</application>
	<application>junos-ntp</application>
	
	<application>junos-ssh</application>

	</match> </policy> </policy> </policies> </security> </configuration>
// Add elements under 'match' $match->append("source-address","any"); $match->append("destination-address","any");	<configuration> <security> <policies> <policy> <from-zone-name>trust</from-zone-name> <to-zone-name>untrust</to-zone-name> <policy> <name>my-sec-policy</name> <match> <application>junos-ftp</application> <application>junos-ntp</application> <application>junos-ssh</application> <source-address>any</source-address> <destination-address>any</destination-address> </match> </policy> </policy> </policies> </security> </configuration>
// Add the 'then' hierarchy with a child 'permit' element $policyOne->append("then")->append("permit");	<configuration> <security> <policies> <policy> <from-zone-name>trust</from-zone-name> <to-zone-name>untrust</to-zone-name> <policy> <name>my-sec-policy</name> <match> <application>junos-ftp</application> <application>junos-ntp</application> <application>junos-ssh</application> <source-address>any</source-address> <destination-address>any</destination-address> </match> <then> <permit/> </then> </policy> </policy> </policies> </security> </configuration>
// Complete code and final configuration	<configuration>

```
 $builder = new XMLBuilder();
 $policy =
 $builder->create_new_config
("security","policies","policy");
 $policy->append("from-zone-
                        name","trust");
$policy->append("to-zone-
                      name","untrust");
$policyOne = $policy->append("policy");
$policyOne->append("name",”my-sec-
                                policy");
$match = $policyOne->append("match");
$match->append("application",
        "junos-ftp,junos-ntp,junos-ssh");
$match->append("source-address","any");
$match->append("destination-
                         address","any");
$policyOne->append("then")->
                        append("permit");
```

```
<security>
<policies>
<policy>
<from-zone-name>trust</from-zone-name>
<to-zone-name>untrust</to-zone-name>
<policy>
<name>my-sec-policy</name>
<match>
<application>junos-ftp</application>
<application>junos-ntp</application>
<application>junos-ssh</application>
<source-address>any</source-address>
<destination-address>any</destination-
address>
</match>
<then>
<permit/>
</then>
</policy>
</policy>
</policies>
</security>
</configuration>
```

Related Documents

- NETCONF PHP Toolkit Class: Device
- NETCONF PHP Toolkit Class: XML
- NETCONF PHP Toolkit Class: XMLBuilder

CHAPTER 4

NETCONF PHP Toolkit Program Files

- Creating NETCONF PHP Toolkit Program Files
- Parsing RPC Reply
- Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program

Creating NETCONF PHP Toolkit Program Files

- Creating and Executing NETCONF PHP Toolkit Program File
- Example: Using the NETCONF PHP Toolkit to execute an Operational Request RPC

Creating and Executing a NETCONF PHP Toolkit Program File

You can use the NETCONF PHP toolkit to connect to a device, open a NETCONF session, and create and execute operational and configuration requests. After installing the NETCONF PHP toolkit, which is described in "Downloading and Installing the NETCONF PHP Toolkit" , the general procedure is:

1. Creates a PHP program that includes code to connect to a deviceand to execute the desired operations or request.
2. Execute the php Program.

These steps are reviewed in detailin the following sections.

- Creating a NETCONF PHP Toolkit Program file
- Compiling and Executing a NETCONF PHP Toolkit Program File

Creating a NETCONF PHP Toolkit program

NETCONF PHP Toolkit programs have same generic framework. To create a basic NETCONF PHP toolkit program:

1. Create a .php file.

2. Include Device.php class in your program.

```
include ('netconf/Device.php ');
```

3. Create a device object and call `connect()` method
   This also creates NETCONF session with the NETCONF server over SSHv2

```
$device_name = new Device ("hostname", "username",
"password"); // if using password based authentication
$device_name = new Device ("hostname", "username");
        // if using public-private key based authentication
$device_name->connect ();
```

4. Execute operational and configuration requests by executing RPCs and performing NETCONF operations on the Device object.

For example, to execute an operational request to retrieve alarm information from the device, include the following line of code:

```
$reply = device->execute_rpc("get-alarm-information");
```

5. To print, parse or take action on rpc replies received from NETCONF server code to print rpc reply in XML, in string format
   ```
   echo $reply->to_string();
   ```
6. Close the device and release resources by calling the close() method on the Device object.
   ```
   $device_name->close();
   ```

Sample Netconf

The following sample code illustrates a simple NETCONF PHP toolkit program, show_alarm.php, which connects to a device and executes an operational request for alarm information.

```
// show_alarm.php
<?php
include(netconf/Device.php)

   //create a device object and establish a NETCONF
    session
$d= new Device("hostname", "username", "password");
$d->connect();

//send and receive rpc reply as xml
$reply= $d->execute_rpc("get-alarm-information");

//print the rpc reply and close the device
echo $reply->to_string();
$d->close();
?>
```

Executing a NETCONF PHP Toolkit program file

To execute a NETCONF php program, run the program from configuration management server.

```
php show_alarm.php
```

Example: Using NETCONF PHP Toolkit to Execute an operational Request RPC

This NETCONF PHP toolkit program executes an RPC to obtain operational information from a device, which is then printed to standard output. This example serves as an instructional example for creating and executing a basic NETCONF PHP toolkit program.

- Requirements
- Overview
- Configuration
- Verification
- Troubleshooting

Requirements

- NETCONF PHP toolkit is installed on the configuration management server.
- Client application can log in to the device where the NETCONF server resides.
- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

Overview

You can use the NETCONF PHP toolkit to request operational information from a remote device. The following example illustrates how to create a NETCONF PHP toolkit program to execute an operational request from the Junos XML API on a device running Junos OS. You can also execute operation request by using function that takes cli command.
The example also explains how to execute the program, and verify the results.

Configuration

Creating the PHP program

Step-by-step procedure

To construct the php program file that contain the code for operational request

1. Give the file a descriptive name.
   For example `Get_Alarm_Info.php`
2. Include the appropriate files according to need, present in
   For Example
   ```
   include('netconf/Device.php');
   ```
3. Create a device object and call the `connect()` method.

This creates a NETCONF session over SSHv2 with the NETCONF server.
You must update the code with the appropriate arguments for connection to and authentication on your specific device.

```
$d = new Device("10.10.1.1", "admin", "PaSsWoRd", null);
$d->connect();
```

Having established a Device object, you can perform NETCONF operations on the device.

4. Call the `executeRPC()` method with the operational request RPC command as the argument.

   This example uses the Junos XML API get-alarm-information RPC. The reply, which is returned in XML, is stored in the rpc_reply variable.
   ```
   $reply = $d->execute_rpc("get-alarm-information");
   ```

5. Add code to take action on the RPC reply.
   The following code converts the NETCONF server's reply to a string and prints it to the screen:
   ```
   echo $replt->to_string();
   ```

6. Close the device and release resources by calling the close() method on the device object.
   ```
   $d->close();
   ```

Result: The Complete Program

```
// Get_Alarm_Info.php
<?php
include(netconf/Device.php)

// create a device object and establish a NETCONF session

$d= new Device("hostname", "username", "password");
$d->connect();

//send and receive rpc reply as xml
$reply= $d->execute_rpc("get-alarm-information");

//print the rpc reply and close the device
echo $reply->to_string();
$d->close();
?>
```

Running PHP Program

Execute the resulting program

```
php Get_Alarm_Info.php
```

Verification

Verifying program execution

Purpose: Verify that the Get_Alarm_Info runs correctly

Action: If the program executes successfully, it establishes a connection and creates a NETCONF session with the specified device. The program sends the get-alarm-information RPC tag to the NETCONF server, and the server responds with the requested operational information enclosed in the <rpc-reply> tag element. The program prints the reply to standard out.

Following is a sample RPC reply with some output omitted for brevity.

<rpc-reply xmlns="urn:ietfaramsml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/13.2I0/junos">
<alarm-information xmlns="http://xml.juniper.net/junos/13.2I0/junos-alarm">
<alarm-summary>
……..
output omitted
……..

</alarm-summary>
</alarm-information>
</rpc-reply>

Parsing an RPC Reply

After submitting an operational or configuration request to the NETCONF server, the server responds with an RPC reply.

$reply=$device_name->execute_rpc("get-system-information");

There are two approaches to parse an XML reply within the context of NETCONF PHP Toolkit

1. Use the parsing methods available in standard php class libraries (DOMDocument, DOMXPath) for Document Object Model(DOM).
2. Use find_value() method available in XML class.

Using DOM class libraries

To parse XML rpc reply do the following:
1. Call the get_owner_document( ) on the reply object to return the Document object.
2. Create a new DOMXPath object for document object.
3. Use query( ) method to go to the desired path and print the desired value.

Using find_value( ) method available in XML class

To parse XML rpc reply do the following:
1. Make an array mentioning hierarchy
   For example,
   If you want to know os-name, for below hierarchy

```
<rpc-reply xmlns…….>
<system-information>
<hardware-model> ……..</hardware-model>
<os-name>junos</os-name>
<os-version>……..</os-version>
<host-name>……..</host-name>
</system-information>
</rpc-reply>
```

   then make array of system-information, os-name
   $list= array("system-information","os-name");

2. Call the find_value( ) on xml object returned by execute_rpc( ) method with argument as $list.
   $val=$reply->find_value($list);

Complete Code

Following example prints the OS-name returned by server in rpc-reply

```
<?php
include('netconf/Device.php');

//creating a new device and establishing netconf session
$d= new Device("10.209.16.204", "regress", "MaRtIni");
$d->connect();
echo "connected to device";

// getting system information from server by using execute_rpc()
try
{
$inven=$d->execute_rpc("get-system-information");
echo "rpc reply\n\n";

//rpc reply from server is in xml form, so convert xml into
string and print
$str=$inven->to_string();
echo $str;

//one way is by using xml method find_value()
$list = array('system-information','os-name');
$val=$inven->find_value($list);
echo "\n\nvalue is".$val ."\n";

//other is by using php classes for XML

//get the owner document from xml object
$xmldoc = $inven->get_owner_document();

//define DOMXPath object and use query() to parse rpc-reply
$xpathvar = new Domxpath($xmldoc);

//use local-name() to match on element name with or without
namespaces
$queryResult = $xpathvar->query('/*[local-name()="rpc-
reply"]/*[local-name()="system-information"]/*[local-name()="os-
name"]');
echo "\n node value  : ";
foreach($queryResult as $result){
                echo $result->textContent;
       }
}
catch(Exception $e)
{
echo 'exception', $e->getMessage(), "\n";
```

```
}

//closing device
$d->close();
echo "device closed";
?>
```

Running PHP Program

Execute the resulting program

```
php parse_rpc_reply.php
```

Verification

Verifying program execution

Purpose: Verify that the parse_rpc_reply.php runs correctly

Action Following will the reply from server

```
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietfaramsml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/13.2I0/junos">
<system-information>
<hardware-model> ……..</hardware-model>
<os-name>junos</os-name>
<os-version>……..</os-version>
<host-name>……..</host-name>
</system-information>
</rpc-reply>
```

So when you print os-name, it should return Junos

Troubleshooting

Troubleshooting NETCONF Exceptions

Problem: A NETCONF exception occurs, and you see the following error message:

Exception in thread "main" net.juniper.netconf.NetconfException: There was a problem while connecting to 10.10.1.1:830 …(or some other host name)

NETCONF over SSH might not be enabled on the device where the NETCONF server resides, or it might be enabled on a different port.

Solution: Ensure that you have enabled NETCONF over SSH on the device where the NETCONF server resides.Since the example program does not specify a specific port number in the Device arguments, the NETCONF session is established on the default NETCONF-over-SSH port, 830. To verify whether NETCONF over SSH is enabled on

the default port for a device running Junos OS, enter the following operational mode command on the remote device:

```
user@host> show configuration system services
ftp;
   netconf {
                  ssh;
                  }
```

If the netconf configuration hierarchy is absent, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

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

If the netconf configuration hierarchy specifies a port other than the default port, include the new port number in the Device object constructor arguments. For example, the following device is configured for NETCONF over SSH on port 12345:

```
user@host> show configuration system services
     netconf {
           ssh {
              port 12345;
               }
              }
```

To correct the connection issue, include the new port number in the Device arguments.

```
$device_name = new Device("10.10.1.1", "admin",
                    "PaSsWoRd",null,12345)
```

Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program

The following sections outline exception errors that you might encounter when executing a NETCONF PHP toolkit program. These sections also present potential causes and solutions for each error.

Troubleshooting Connection Errors: No Connection
- Trouble shooting authentication Errors
- Troubleshooting NETCONF session errors

Troubleshooting Connection Errors: No Connection

Problem: An IllegalStateException exception occurs, and you see the following error message:

Cannot execute rpc, need to establish a connection first

Cause: An SSHv2 connection or NETCONF session was not established with the remote device.

Solution: Call the connect() method on the device object to establish an SSHv2 connection and a NETCONF with the device on which the NETCONF server runs. Once the connection and session are established, RPC execution should be successful.

Troubleshooting Authentication Errors

Problem: A NETCONF exception occurs, and you see the following error message (sometimes program exits without running anything):

Could not connect to device Authentication Failed

or

Wrong Username or password

or

Device not found

Cause: An error message for failed authentication could have several possible causes, including the following:

- The host or authentication details passed as arguments to the Device constructor are incorrectly entered in the program code.
- The arguments for the Device object are correct, but there is no corresponding user account created on the device to which you are connecting.

Solution: If there is no user account on the device to which you are connecting, create the account with the appropriate authentication. For more information about configuring user accounts on a device running Junos OS, see the Junos OS System Basics Configuration Guide.

If the user account exists on the remote device, but the arguments for the Device constructor are entered incorrectly in the program code, correct the arguments and recompile the program.

Troubleshooting NETCONF Session Errors

Problem: A NETCONF exception occurs, and you see the following error message:

There was a problem while connecting to 10.100.0.1:830

Cause: NETCONF over SSH might not be enabled on the device where the NETCONF server resides, or it might be enabled on a different port.

Solution: Ensure that you have enabled NETCONF over SSH on the device where the NETCONF server resides. If your NETCONF PHP toolkit program does not specify a specific port number in the Device argurments, the NETCONF session is established on the default NETCONF-over-SSH port, 830. To verify whether NETCONF over SSH is enabled on the default port for a device running Junos OS, enter the following operational mode command on the remote device:

```
user@host> show configuration system services
        ftp;
           netconf {
                    ssh;
```

```
}
```

If the netconf configuration hierarchy is absent, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

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

If the netconf configuration hierarchy is absent, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

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

If the netconf configuration hierarchy specifies a port other than the default port, you should include the new port number in the Device object constructor arguments. For example, the following device is configured for NETCONF over SSH on port 12345:

```
user@host> show configuration system services
netconf {
      ssh {
            port 12345;}}
```

To correct the connection issue, include the new port number in the Device arguments.

```
$device_name = new Device("10.10.1.1", "admin",
                          "PaSsWoRd", null, 12345);
```

Related Documentation

- Creating and Executing a NETCONF PHP Toolkit Program File
- NETCONF PHP Toolkit Class: Device
- NETCONF PHP Toolkit Overview

Chapter 5

Performing Operational and Configuration Tasks

- Using the NETCONF PHP Toolkit to perform Operational Tasks
- Using the NETCONF PHP Toolkit to perform Configuration Tasks

Using NETCONF PHP Toolkit to perform Operational Tasks

- Using Device object methods to Execute RPCs and Operational Commands
- Example: Using NETCONF PHP Toolkit to execute cli commands

Using Device Object Methods to Execute RPCs and Operational Commands

The NETCONF PHP toolkit Device object has methods to request information from and perform operational tasks on remote devices. When appropriate, the methods are overloaded to take a number of different formats.

- Executing RPCs
- Executing Operational Mode Commands

Executing RPCs

To execute a remote procedure call (RPC), call the `execute_rpc()` method on the Device object. This function sends RPC (as XML object or as string) over the default NETCONF session and get the response as an XML object.

The syntax to call this method is

```
$device_name->execute_rpc(string RPC content)
$device_name->execute_rpc(XML RPC content)
```

The following code snippet executes the Junos XML API get-system-information RPC using a string argument

```
$device_name = new Device("10.100.0.1", "ADMIN" , "PaSSwd",
null);
$device_name->connect();
try {
$rpc_reply = $device_name->execute_rpc("get-system-
information");
echo $rpc_reply->to_string();
}
Catch(Exception $e)
{
echo "exception".$e->getMessage();
//additional processing for exception
```

```
}
$device_name->close();
```

Executing Operational Mode Commands

To execute an operational mode command to request information from or perform operational tasks on a device running Junos OS, call the `run_cli_command ()` method in Device class. The `run_cli_command ()` method sends a Junos OS operational mode command to the NETCONF server on the remote device. The argument is a string representing the operational mode command that you would enter in the Junos OS CLI. The RPC is processed by the NETCONF server, which returns the RPC reply. Starting with Junos OS Release 11.4, the return string is the same ASCII-formatted output that you see in the Junos OS CLI. For devices running earlier versions of Junos OS, the return string contains Junos XML tag elements.

The syntax to run this method is

```
$device_name->run_cli_command(string command)
```

The following code snippet sends the CLI operational mode command show system services to the NETCONF server on a device running Junos OS:

```
$device_name = new Device("10.100.0.1", "ADMIN" , "PaSSwd",
null);
$device_name->connect();
try {
$rpc_reply = $device_name->run_cli_command("show system
services");
echo $rpc_reply->to_string();
}
Catch(Exception $e)
{
echo "exception".$e->getMessage();
//additional processing for exception
}
$device_name->close();
```

Example: Using NETCONF PHP Toolkit to Execute CLI commands

This NETCONF PHP Toolkit program demonstrate the run_cli_command() method, which sends the specified Junos OS operational mode command to the NETCONF server to request information from or perform operational tasks on a device running Junos OS.

- Requirements
- Overview
- Configuration
- Verification

Requirements

- Routing, switching, or security device running Junos OS.

- NETCONF PHP toolkit is installed on the configuration management server.
- Client application can log in to the device where the NETCONF server resides.
- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

Overview

The NETCONF PHP toolkit Device class contains the `run_cli_command()` method, which takes a Junos OS CLI operational mode command and converts it to an equivalent RPC in XML that can be processed by the NETCONF server. The `run_cli_command()` method takes as an argument the string representing an operational mode command that you enter in the Junos OS CLI.

The following example executes the show system services command on a device running Junos OS. The return value for the method is a string. Starting with Junos OS Release11.4, the return string is the same ASCII-formatted output that you see in the Junos OS CLI. For devices running earlier versions of Junos OS, the return string contains Junos XML tag elements.

Configuration

Step by step Procedure

Create a php program

To construct the php program file

1. Give the file a descriptive name like ExecuteCliCommand.php.
2. Add the code to the file and update the environment-specific variables such as the remote host IP address, username, password, and <rpc-reply> tag elements.

```
// The complete code for ExecuteCliCommand.php is
   presented here:
    <?php
     include("netconf/Device.php");


    $device_name = new Device("10.100.0.1", "ADMIN",
                                    "PaSSwd", null);
    $device_name->connect();


    try {
    $rpc_reply = $device_name->run_cli_command("show
                                    system services");
    echo $rpc_reply->to_string();
        }

    Catch(Exception $e)
        {
     echo "exception".$e->getMessage();
     //additional processing for exception
        }

    $device_name->close();
    ?>
```

Running PHP file

Execute the php file, ExecuteCliCommand.php by

```
php ExecuteCliCommand.php
```

Verification

Purpose: Verify that the ExecuteCliCommand program runs properly

Action: If the program executes successfully, it establishes a connection and creates a NETCONF session with the specified device. The program converts the Junos OS CLI operational mode command show system services to an RPC and sends the RPC to the NETCONF server. The server responds with the requested operational information enclosed in the <rpc-reply> tag element The program parses the RPC reply and prints the resulting system services.The following sample output is from a Juniper Networks router.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/13.2I0/junos">
<configuration-information>
<configuration-output>
ftp;
ssh {
   root-login allow;
}
telnet;
xnm-clear-text;
netconf {
   ssh;
}
web-management {
   http;
}
</configuration-output>
</configuration-information>
</rpc-reply>
```

Related Documentation

- Example: Using the NETCONF PHP Toolkit to Execute an Operational Request RPC
- Example: Using the NETCONF PHP Toolkit to Load and Commit a Configuration
- Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program
- NETCONF PHP Toolkit Class: Device
- NETCONF PHP Toolkit Overview

Using the NETCONF PHP Toolkit to perform Configuration Tasks

- Using Device Object Methods to Load Configuration Changes
- Example: Using the NETCONF PHP Toolkit to Load and Commit a Configuration
- Example: Using the NETCONF PHP Toolkit to Load Set Configuration Commands

Using Device Object Methods to Load Configuration Changes

The NETCONF PHP toolkit Device object has methods to help you configure remote devices. When appropriate, the methods are overloaded to take a number of different formats.

To load configuration data on a remote device, the Device object has several methods that enable you to define the configuration data as a set of Junos OS configuration mode commands, formatted ASCII text, or Junos XML tag elements. You can supply the configuration data in the program code, or you can reference data files that include the desired configuration changes.

To configure a **private copy** of the candidate configuration, call the `open_configuration()` method on the device object before loading your configuration changes. This is equivalent to the configure private command in the Junos OS CLI. If you omit the call to the `open_configuration()` method, your configuration changes are loaded into the global copy of the candidate configuration.

The method used to load the configuration data depends on the source and the format of the data. In the following methods, the string argument *loadType* has a value of either merge or replace, which performs the equivalent of the configuration mode commands load merge or load replace on a device running Junos OS.

- ***Junos OS configuration mode commands***—The following methods load configuration data as a set of Junos OS configuration mode commands. These methods are only supported on devices running Junos OS Release 11.4 or a later release. Junos OS executes the configuration instructions line by line. For each element, you can specify the complete statement path in the command, or you can use navigation commands, such as edit, to move around the configuration hierarchy as you do in CLI configuration mode.

  `load_set_configuration(String setCommands)`—Specify the configuration data in the program code, either as a method argument or as a variable passed to the method.
  `load_set_file(String filePath)`—Load the configuration data from the file specified by *filePath*.

- ***Formatted ASCII text***—The following methods load configuration data as formatted ASCII text. Use the standard Junos OS CLI notations—the newline character, tabs, spaces, braces, and square brackets—to indicate the hierarchical relationships between configuration statements.

`load_text_configuration(String textConfiguration, String loadType)` — Specify the configuration data in the program code, either as a method argument or as a variable passed to the method.
`Load_text_file(String filePath, String loadType)` — Load the configuration data from the file specified by *filePath*.

- ***Junos XML tag elements***—The following methods load configuration data as Junos XML tag elements. Include the tag elements representing all levels of the configuration hierarchy under the root, the <configuration> tag element, down to each new or changed element.

  `loadXMLConfiguration(String XMLConfiguration, String loadType)` — Specify the configuration data in the program code as XML object, which is passed to the method.
  `load_xml_file(String filePath, String loadType)` — Load the configuration data from the file specified by *filePath.*

  The following code snippet merges the ftp statement into the candidate configuration at the [edit system services] hierarchy level. The PHP statement for each type of load configuration method is shown. When loading from a file, the file should contain the appropriate hierarchy in the desired format.

  ** config_set.txt*

```
set system services ftp
```

  config_text.txt

```
system
   {
     services
     {
      ftp;
      }
     }
```

  Config_xml.txt

```
<system>
    <services>
         <ftp/>
     </services>
</system>
```

```
$config_file_set= "configs\config_set.txt";
$config_file_text= "configs\config_text.txt";
$config_file_xml = "configs\config_xml.txt"

$device_name = new Device("10.100.0.1", "AdMiN", "Passwd",null);
$device_name->connect();

//open a private copy of candidate configuration
$device_name->open_configuration();

//load configuration data as Junos OS configuration mode commands
$device_name->load_set_configuration("set system services ftp");
$device_name->load_set_file($config_file_set);

//load configuration data as formatted ASCII text
```

```
$device_name->load_text_configuration("system { services { ftp;
}}", "merge");
$device_name->load_text_file($config_file_text, "merge");

//load configuration data as Junos XML tag elements
$device_name->load_xml_configuration ("<system> <services> <ftp/>
</services> </system>" ,  "merge")
$device_name->load_xml_file($config_file_xml, "merge");

$device_name->commit();
$device_name->close();
?>
```

Example: Using the NETCONF PHP Toolkit to Load and commit a configuration

The following example NETCONF PHP toolkit program constructs a configuration hierarchy, which is then merged with the candidate configuration on the specified device. The resulting configuration is then committed. The sample configuration hierarchy is for a device running Junos OS.

- Requirements
- Overview
- Configuration
- Verification
- Troubleshooting

Requirements

- Routing, switching, or security device running Junos OS.
- NETCONF PHP toolkit is installed on the configuration management server.
- Client application can log in to the device where the NETCONF server resides.
- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

Overview

The following example performs a load merge operation to update the candidate configuration on a device running Junos OS and then commits the new configuration.The XML hierarchy that will be added into the configuration can be constructed with the XMLBuilder object and stored in the ftp_config variable or directly passed as string. Alternatively, you can load configuration data as text and, for devices running Junos OS Release 11.4 or a later release,as a set of Junos OS configuration mode commands.

The new configuration hierarchy, which enables FTP service on the device, is:

```
<configuration>
   <system>
           <services>
              <ftp/>
            </services>
```

```
        </system>
    </configuration>
```

The program code creates a new Device object and calls the `connect()` method. This establishes an SSHv2 connection and a default NETCONF session with the device on which the NETCONF server runs.

To prevent conflicts with other users who might simultaneously edit the candidate configuration, the code calls the `lock_config()` method on the device object to lock the configuration. If the lock fails, the method generates an error message, and the program exits. If the lock is successful, the `load_xml_configuration()` method loads the new configuration hierarchy into the candidate configuration using the merge option.

Once the new configuration hierarchy is merged with the candidate configuration, the program attempts to commit the configuration. If the commit operation is unsuccessful, the program prints the associated error message. The program then unlocks the configuration and closes the NETCONF session and device connection.

Configuration

Creating the php program file

Step by step Procedure

To construct the PHP Program file that contains code for configuration change and requests:

1. Give the file a descriptive name.
   Example `EditConfig.php`

2. Add the code to the file and update the environment-specific variables such as the remote host IP address, username, and password.

```
// The complete PHP code for the EditConfig.php program is
presented here.

    <?php
    include("netconf/Device.php");

    $xml_builder= new XMLBuilder();
    $ftp_config =$xml_builder-
    >create_new_config("system","services","ftp");

    // create the device and establish NETCONF session
    $device_name = new Device("10.100.0.1", "AdMiN",
    "Passwd",null);
    $device_name->connect();

    //lock the configuration
    $isloc= $device_name->lock_config();

    // if locked successfully then only perform any action
    if($isloc)
```

```
{

//load and commit the configuration
$device_name->load_xml_configuration($ftp_config-
>to_string(), "merge");
$device_name->commit();
}

else
{
echo "could not lock the configuration";
exit(1);
}

//Unlock the configuration and close the device
$device_name->unlock_config();
$device_name->close();

?>
```

Running the PHP Program

Run the above code EditConfig.php by :

```
php EditConfig.php
```

Verification

Verifying Program Execution

Purpose: Verify that the EditConfig program runs correctly.

Action: If the program executes successfully, it establishes a connection and a creates a NETCONF session with the specified device. The program merges the new hierarchy with the candidate configuration on the device and commits the configuration.

You can verify that the configuration was correctly merged and committed by viewing the resulting configuration on the remote device. The ftp statement should now be in the active configuration. On a device running Junos OS, enter the following operational mode command to view the [edit system services] hierarchy:

```
user@host> show configuration system services
ftp;
netconf {
ssh;
}
```

Troubleshooting

Troubleshooting Error Messages

Troubleshooting Error Messages

Problem: The following error message is printed to the display:

```
Could not lock configuration. Exit now.
```

Solution: Another user currently has a lock on the candidate configuration. Wait until the lock is released and execute the program.

Example: Using the NETCONF PHP Toolkit to Load Set Configuration Commands

NETCONF PHP toolkit program demonstrates the load_set_configuration() method, which updates the configuration using a set of Junos OS configuration mode commands.

- Requirements
- Overview
- Configuration
- Verification

Requirements

- Routing, switching, or security device running Junos OS Release 11.4 or later.
- NETCONF PHP toolkit is installed on the configuration management server.
- Client application can log in to the device where the NETCONF server resides.
- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

Overview

The Device class contains the `load_set_configuration()` and `load_set_file()` methods, which load configuration data as a set of Junos OS configuration mode commands on devices running Junos OS Release 11.4 or a later release. For each configuration element, you can specify the complete statement path in the command, or you can use navigation commands, such as edit, to move around the configuration hierarchy as you do in CLI configuration mode. The NETCONF PHP toolkit converts the command set to the equivalent RPC in XML that can be processed by the NETCONF server on devices running Junos OS. Junos OS executes the configuration instructions line by line.

The syntax to use method is:

```
$device_name->load_set_configuration (String setCommands);
$device_name->load_set_file (String filepath)
```

The `load_set_configuration()` method takes as an argument the configuration command string that you would enter in Junos OS CLI configuration mode. For example, to add the ftp statement at the [edit system services] hierarchy level, you use the set system services ftp command. The `load_set_file()` method takes as an argument the path of the file containing the set of configuration commands. You can also use both methods to load multiple commands. To load multiple commands using the `load_set_configuration()` method, you can either list the commands as a single string and separate them with the \n newline sequence, or you can execute the method separately for each command. To load

multiple commands using the `load_set_file()` method, place each command on a separate line in the file.

The program in this example loads two configuration commands, which merge two statements into the candidate configuration on a device running Junos OS Release 11.4.The first command, set system services ftp, adds the ftp statement at the [edit system services] hierarchy level. The second command, set interfaces ge-0/0/0 disable, adds the disable statement at the [edit interfaces ge-0/0/0] hierarchy level. The relevant statements in the program code are:

```
$system_config = "set system services ftp";
$interfaces_config = "set interfaces ge-0/0/0 disable";
$device_name->load_set_configuration($system_config);
$device_name->load_set_configuration($interfaces_config);
```

Configuration

Creating a PHP Program

Step by Step Procedure

To construct the PHP Program file

1. Give the file a descriptive name
   Example: `LoadSetConfig.php`.

2. Add the code to the file and update the environment-specific variables such as the remote host IP address, username, password.

The complete PHP code for the LoadSetConfig.php program is presented here.
If you load the set of commands from a file, create a file containing the commands, and replace the two load_set_Configuration() method calls with a call to the load_set_file() method.

```
 <?php
include("netconf/Device.php");

$system_config = "set system services ftp";
$interfaces_config = "set interfaces ge-0/0/0
                                                    disable";

// create the device and establish NETCONF session
$device_name = new Device("10.100.0.1", "AdMiN",
                                                "Passwd",null);
$device_name->connect();

//lock the configuration
$isloc= $device_name->lock_config();

// if locked successfully then only perform any action
if($isloc)
{
//load and commit the configuration
$device_name->load_set_configuration($system_config");
$device_name->load_set_configuration($interfaces_config");
$device_name->commit();
```

```
}

else
{
//if can not lock successfully then exit
echo "could not lock the configuration";
exit(1);
}

//Unlock the configuration and close the device
$device_name->unlock_config();
$device_name->close();

?>
```

Running PHP Code

To execute LoadSetConfig.php use the following command

```
php LoadSetConfig.php
```

Verification

To confirm that the program is working properly:
- Verifying Program Execution
- Verifying the Configuration Changes
- Verifying the Commit

Verifying Program Execution

Purpose: Verify that the LoadSetConfig.php program runs correctly.

Action: If the program executes successfully, it establishes a connection and creates a NETCONF session with the specified device. The program merges the new statements with the candidate configuration on the device and commits the configuration.

```
Connection successful.
Configuration successfully locked.
Updating configuration.
Committing configuration.
```

Verifying the Configuration Changes

Purpose: You can verify that the configuration was correctly merged and committed by viewing the resulting configuration on the remote device. The ftp and the disable statements should now be in the active configuration. On a device running Junos OS,

issue the following operational mode commands to view the [edit system services] and [edit interfaces] hierarchy levels:

Action:

```
admin@host> show configuration system services
      ftp;
      netconf
       {
       ssh;
       }
      admin@host> show configuration interfaces
      ge-0/0/0 {
      disable;
      }
```

Verifying the commit

Purpose: Additionally, you can review the commit log to verify that the commit was successful.On a device running Junos OS, issue the show system commit operational mode command to view the commit log. In this example, the log confirms that the user admin committed the candidate configuration in a NETCONF session at the given date and time.

Action: Issue the show system commit operational mode command and review the commit log.

```
admin@host> show system commit
0 2014-20-02 14:16:44 PDT by admin via netconf
1 2014-20-08 14:33:46 PDT by root via other
```

Related Documentation

- Example: Using the NETCONF PHP Toolkit to Execute an Operational Request RPC
- Example: Using the NETCONF PHP Toolkit to Load and Commit a Configuration
- Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program
- NETCONF PHP Toolkit Class: Device
- NETCONF PHP Toolkit Overview

Chapter 6

NETCONF PHP Toolkit Example

- Example: Using the NETCONF PHP Toolkit to Execute an Operational Request RPC

Example: Using the NETCONF PHP Toolkit to Execute an Operational Request RPC

This NETCONF PHP toolkit program executes an RPC to obtain operational information from a device, which is then printed to standard output. This example serves as an instructional example for creating and executing a basic NETCONF PHP toolkit program.

- Requirements
- Overview
- Configuration
- Verification
- Troubleshooting

Requirements:

- NETCONF PHP toolkit is installed on the configuration management server.
- Client application can log in to the device where the NETCONF server resides.
- NETCONF service over SSH is enabled on the device where the NETCONF server resides.

Overview

You can use the NETCONF PHP toolkit to request operational information from a remote device. The following example illustrates how to create a NETCONF PHP toolkit program to execute an operational request from the Junos XML API on a device running Junos OS. You can also execute operation request by using function that takes cli command.

The example also explains how to execute the program, and verify the results.

Configuration

Creating the PHP program

Step-by-step procedure

To construct the php program file that contain the code for operational request

1. Give the file a descriptive name.
   For example `Get_Alarm_Info.php`

2. Include the appropriate files according to need, present in
   `For Example`
   `include('netconf/Device.php');`

3. Create a device object and call the `connect()` method.

This creates a NETCONF session over SSHv2 with the NETCONF server. You must update the code with the appropriate arguments for connection to and authentication on your specific device.

```
$d = new Device("10.10.1.1", "admin", "PaSsWoRd",
                                                    null);
$d->connect();
```

Having established a Device object, you can perform NETCONF operations on the device.

4. Call the `executeRPC()` method with the operational request RPC command as the argument.

   This example uses the Junos XML API get-alarm-information RPC. The reply, which is returned in XML, is stored in the rpc_reply variable.

   ```
   $reply = $d->execute_rpc("get-alarm-information");
   ```

5. Add code to take action on the RPC reply.The following code converts the NETCONF server's reply to a string and prints it to the screen:

   ```
   echo $reply->to_string();
   ```

6. Close the device and release resources by calling the close() method on the device object.

   ```
   $d->close();
   ```

Result: The Complete Program

```
// Get_Alarm_Info.php
<?php
include(netconf/Device.php)

// create a device object and establish a NETCONF
                                                session
$d= new Device("hostname", "username", "password");
$d->connect();

//send and receive rpc reply as xml
$reply= $d->execute_rpc("get-alarm-information");

//print the rpc reply and close the device
echo $reply->to_string();
$d->close();
?>
```

Running PHP Program

Execute the resulting program

```
php Get_Alarm_Info.php
```

Verification

Verifying program execution

Purpose: Verify that the Get_Alarm_Info runs correctly

Action: If the program executes successfully, it establishes a connection and a creates a NETCONF session with the specified device. The program sends the get-alarm-information RPC to the NETCONF server, and the server responds with the requested operational information enclosed in the <rpc-reply> tag element. The program prints the reply to standard out.

Following is a sample RPC reply with some output omitted for brevity.

<rpc-reply xmlns="urn:ietfaramsml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/13.2I0/junos">
<alarm-information xmlns="http://xml.juniper.net/junos/13.2I0/junos-alarm">
<alarm-summary>
……..
output omitted
……..
</alarm-summary>
</alarm-information>
</rpc-reply>

Troubleshooting NETCONF Session Errors

Problem: A NETCONF exception occurs, and you see the following error message:
`There was a problem while connecting to 10.100.0.1:830`

Cause: NETCONF over SSH might not be enabled on the device where the NETCONF server resides, or it might be enabled on a different port.

Solution: Ensure that you have enabled NETCONF over SSH on the device where the NETCONF server resides. If your NETCONF PHP toolkit program does not specify a specific port number in the Device argurments, the NETCONF session is established on the default NETCONF-over-SSH port, 830. To verify whether NETCONF over SSH is enabled on the default port for a device running Junos OS, enter the following operational mode command on the remote device:

```
user@host> show configuration system services
ftp;
      netconf {
            ssh;
               }
```

If the netconf configuration hierarchy is absent, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

```
[edit]
```

```
user@host# set system services netconf ssh
user@host# commit
```

If the netconf configuration hierarchy is absent, issue the following statements in configuration mode to enable NETCONF over SSH on the default port:

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

If the netconf configuration hierarchy specifies a port other than the default port, you
should include the new port number in the Device object constructor arguments. For
example, the following device is configured for NETCONF over SSH on port 12345:

```
user@host> show configuration system services
netconf {
ssh {
port 12345;
}
```

To correct the connection issue, include the new port number in the Device arguments.

```
$device_name = new Device("10.10.1.1", "admin", "PaSsWoRd",
                                        null, 12345);
```

Related Documentation

- Example: Using the NETCONF PHP Toolkit to Execute CLI Commands on page
- Example: Using the NETCONF PHP Toolkit to Load and Commit a Configuration on Page
- Example: Using the NETCONF PHP Toolkit to Load Set Configuration Commands
- Troubleshooting Exception Errors in a NETCONF PHP Toolkit Program
- NETCONF PHP Toolkit Overview

FAQs

1. How to establish key based authentication?

- Follow the following steps for establishing public-private key based authentication:

(1) Go to .ssh and generate public-private key pair

```
user@local-host$cd ~/.ssh
user@local-host:~/.ssh$ssh-keygen -t rsa
```

(2) Copy the public key to remote server

```
user@local-host:~/.ssh$ssh-copy-id user@remotehost
```

if you are not inside .ssh then use following command

```
user@local-host$ssh-copy-id –i~/.ssh/id_rsa.pub remotehost
```

(3) Check whether ssh session can be established using public-private authentication

```
user@local-host$ssh user@remotehost
```

2. Troubleshooting errors in sending public key to server

- When no value is passed for option i or when you get error like *no identities found* while copying public key to remote host use *ssh-add* and ssh-agent

```
user@local-host$ ssh-agent $SHELL
user@local-host$ssh-add –L
```

the agent has no identities

```
user@local-host$ssh-add
ssh-rsa.....
.....ssh/id_rsa
user@local-host$ssh-copy-id –i remote-host

// This will add public key to the server
```

3. Problem in including NETCONF API classes like Device.php, XML.php, XMLBuilder.php… etc?

- One way of installing NETCONF PHP API is given in chapter 2, installing NETCONF-PHP-API, other way is by directly downloading APIs from github https://github.com/Juniper/netconf-php and adding it to your desired path. But in that case you have to give complete path in include() for all classes that you intend to use.

4. Understand error in <rpc-reply>

- Sometimes you may get error because you entered wrong rpc tag, did some typing or spelling mistake, so it is important to understand why rpc-error is coming and where is problem in your code.

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

Here:

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

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

<error-severity> - Indicates the severity of the event that caused the NETCONFserver to return the <rpc-error> tag element. The two possible values are error or warning.

<error-info> - Information regarding error.

For example:

An error is returned if an <rpc> is received without a message-id attribute.

```
<rpc xmlns>
<get-config>
<source>
<running/>
</source>
</get-config>
</rpc>
```

```
<rpc-reply>
<rpc-error>
<error-type> rpc</error-type>
<error-tag>missing-attribute</error-tag>
<error-severity>error</error-severity>
<error-info>
    <bad-attribute>message-id<bad-attribute>
     <bad-element>rpc</bad-element>
</error-info>
</rpc-error>
</rpc-reply>
```

For more information, visit rpc-error

5. How to correct the problem that occur while locking a device?

- Sometimes you may get error like:
RPC Failure: configuration database Modified
This error message will occur when a user has acquired an exclusive lock on the configuration database, but the changes have not been committed.
Or your previous program was not executed completely, and exit without commiting and unlocking device.

In order to resolve this issue, go to device CLI, and check to see if there are any users currently logged in who might be modifying the configuration:

```
[edit]
root@test1# run show system users
5:39PM up 14:56, 2 users, load averages: 0.46, 0.48, 0.56

USER   TTY   FROM              LOGIN@    IDLE WHAT
root   p0    172.27.199.119    Tue12PM      - cli
root   p1    172.27.199.127    5:39PM       - cli
```

If no other user is there then either commit your changes or rollback and then again try to lock your device.
For more information visit this link.