Junos® OS

REST API Guide

Release

15.1

Modified: 2015-05-19

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

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Junos® OS REST API Guide
15.1

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.

END USER LICENSE AGREEMENT

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

Table of Contents

List of Tables

About the Documentation

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at http://www.juniper.net/techpubs/.

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

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

Supported Platforms

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

- M Series
- MX Series
- PTX Series
- T Series

Using the Examples in This Manual

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

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

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

Merging a Full Example

To merge a full example, follow these steps:

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

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

   ```
   system {
      scripts {
         commit {
            file ex-script.xsl;
         }
      }
   }
   interfaces {
      fxp0 {
         disable;
         unit 0 {
            family inet {
               address 10.0.0.1/24;
            }
         }
      }
   }
   ```

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

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

Merging a Snippet

To merge a snippet, follow these steps:

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

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

   ```
   commit {
      file ex-script-snippet.xsl; }
   ```

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

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

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

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

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

Documentation Conventions

Table 1 on page ix defines notice icons used in this guide.

Table 1: Notice Icons

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

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

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the **configure** command: user@host> **configure**
`Fixed-width text like this`	Represents output that appears on the terminal screen.	`user@host>` **show chassis alarms** `No alarms currently active`
Italic text like this	• Introduces or emphasizes important new terms. • Identifies guide names. • Identifies RFC and Internet draft titles.	• A policy *term* is a named structure that defines match conditions and actions. • *Junos OS CLI User Guide* • RFC 1997, *BGP Communities Attribute*
Italic text like this	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# **set system domain-name** *domain-name*
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	• To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level. • The console port is labeled **CONSOLE**.
< > (angle brackets)	Encloses optional keywords or variables.	**stub <default-metric** ***metric*>;**
| (pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	**broadcast | multicast** **(*string1* | *string2* | *string3*)**
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	**rsvp { # Required for dynamic MPLS only**
[ ] (square brackets)	Encloses a variable for which you can substitute one or more values.	**community name members [** ***community-ids*** **]**

Indention and braces ({ })	Identifies a level in the configuration hierarchy.	[edit]
		routing-options {
		static {
		route default {
		nexthop *address*;
		retain;
		}
		}
		}

; (semicolon)	Identifies a leaf statement at a configuration hierarchy level.	

GUI Conventions

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

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	• In the Logical Interfaces box, select **All Interfaces**. • To cancel the configuration, click **Cancel**.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select **Protocols>Ospf**.

Documentation Feedback

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

- Online feedback rating system—On any page at the Juniper Networks Technical Documentation site at http://www.juniper.net/techpubs/index.html, simply click the stars to rate the content, and use the pop-up form to provide us with information about your experience. Alternately, you can use the online feedback form at https://www.juniper.net/cgi-bin/docbugreport/.
- 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 JNASC support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

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

Self-Help Online Tools and Resources

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

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

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

Opening a Case with JTAC

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

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

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

CHAPTER 1

Overview

Understanding the REST API

Supported Platforms M Series, MX Series, PTX Series, T Series

The REST API is a Representational State Transfer (REST) interface that enables you to securely connect to Juniper Networks Junos operating system (Junos OS) devices, execute remote procedure calls (**rpc** commands), use a REST API Explorer GUI enabling you to conveniently experiment with any of the REST APIs, and use a variety of formatting and display options, including JavaScript Object Notation (JSON).

The REST API can be configured on Junos OS devices using commands available under the **[edit system services rest]** hierarchy level. Once configured, the REST API becomes available as the **rest** service, a REST-based interface that enables you to submit **rpc** commands to the device from a remote location, and supports GET and POST requests. With the REST API you can:

- Use GET requests to submit **rpc** commands.
- Use POST requests to submit information via **rpc** commands.
- Retrieve configuration information in XML, ASCII (plain text), or JSON.
- Retrieve operational data in XML, ASCII, or JSON.

At the **[edit system services rest]** hierarchy level, you can configure and secure the REST API service on a Junos OS device; set up IP addresses, port numbers, server certificates, control parameters, and trace options; and enable a REST API explorer tool that enables you to try the REST APIs using a convenient GUI.

The following CLI display options are available:

- A **display json** option is added to the | *(pipe)* command. For example, the CLI command **show interfaces | display json** displays the interfaces in JSON notation.
- A **format="json"** option is added to NETCONF server commands to return operational information in JSON notation.

The REST API supports HTTP Basic Authentication, and all requests require a base64-encoded username and password included in the Authorization header. Both HTTP and HTTPS support are available:

- You can use HTTP to exchange content using clear text if you do not need a secure connection.
- We recommend that you use HTTPS to exchange encrypted content using one of the available cipher suites. You can configure the REST API to require server authentication without client authentication, or you can configure mutual authentication.

Once the REST API is configured on the device, new REST endpoints are available for executing either single **rpc** commands via GET or POST requests, or executing multiple **rpc** commands via a single POST request. See "Submitting a GET Request to the REST API" on page 29 and "Submitting a POST Request to the REST API" on page 31 for more information.

The REST API also provides a GUI called the REST API Explorer, which allows you to easily and quickly learn how to use the REST API. It is disabled by default, and can be enabled by specifying **set system services rest enable-explorer**. To learn more about the REST API Explorer, see "Example: Using the REST API Explorer" on page 20.

Related Documentation

- Example: Using the REST API Explorer on page 20
- Configuring the REST API on page 15
- Submitting a GET Request to the REST API on page 29
- Submitting a POST Request to the REST API on page 31
- [edit system services rest] Hierarchy Level on page 35
- *| (pipe)*
- *Pipe (|) Filter Functions in the Junos OS Command-Line Interface*
- *Specifying the Output Format for Operational Information Requests in a NETCONF Session*

CHAPTER 2

Configuring and Using the REST API

Configuring the REST API

Supported Platforms M Series, MX Series, PTX Series, T Series

The REST API can be configured on Junos OS devices using commands available under the **[edit system services rest]** hierarchy level. Once configured, the REST API becomes available as the **rest** service, a REST-based interface that enables you to submit **rpc** commands to the device from a remote location, and supports GET and POST requests.

To enable the REST API on your device, you need to configure:

- Control parameters— These allow you to optionally specify permitted source IP addresses and connection limits common to both HTTP and HTTPS connections.
- REST API Explorer— The REST API provides a GUI called the REST API Explorer, which allows you to easily and quickly learn how to use the REST API. It is disabled by default, and can be enabled by specifying **set system services rest enable-explorer**. To learn more about the REST API Explorer, see "Example: Using the REST API Explorer" on page 20.
- HTTP access— You can specify a list of addresses and TCP ports for incoming connections. HTTP connections are not secure because they exchange credentials and data in clear text, so we recommend using HTTPS.
- HTTPS access (*recommended*)— You can specify a list of addresses and TCP ports for incoming connections, a list of preferred cipher suites, transport layer security (TLS) mutual authentication, and server certificates. HTTPS connections are secure, encrypting both credentials and information.
- Trace options— You can enable tracing for lighttpd, User Interface Script Environment (juise), or both. Trace information for lighttpd is stored at

/var/chroot/rest-api/var/log/lighttpd, and trace information for juise is stored at **/var/chroot/rest-api/var/log/juise**. Tracing is disabled by default.

To configure the optional control parameters for settings common to both HTTP and HTTPS connections:

1. Specify **set system services** rest control allowed-sources **[*value-list*]** to set the permitted IP addresses for both HTTP and HTTPS connections. Use spaces as delimiters between values.
2. Specify **set system services** rest control connection-limit ***limit*** to set the maximum number of allowed simultaneous connections for both HTTP and HTTPS connections. You can assign a value from 1 through 1024 (the default is 64).

To configure HTTP access:

1. Specify **set system services** rest http addresses **[*addresses*]** to set the addresses on which the server listens for incoming HTTP connections.
2. Specify **set system services** rest http port ***port-number*** to set the TCP port for incoming HTTP connections. You can assign a value from 1024 through 65535 (the default is 3000).

To configure HTTPS access:

1. Specify **set system services** rest https addresses **[*addresses*]** to set the addresses on which the server listens for incoming HTTPS connections.
2. Specify **set system services** rest https port ***port-number*** to set the TCP port for incoming HTTPS connections. You can assign a value from 1024 through 65535 (the default is 3443).
3. Specify **set system services** rest https cipher-list**[*cipher-1 cipher-2 cipher-3 ...*]** to configure the set of cipher suites the SSH server can use to perform encryption and decryption functions.
4. Specify **set system services** rest https server-certificate ***local-certificate-identifier*** to configure the server certificate. See *request security pki generate-certificate-request* for information about creating local certificates.
5. You can configure the REST API to require server authentication without client authentication, or you can configure TLS mutual authentication on both the server and client by specifying **set system services** rest https mutual-authentication certificate-authority ***certificate-authority-profile-name***.

To configure trace options for lighttpd, juise, or both, specify **set system services** rest traceoptions **flag *flag*.** Set ***flag*** to **lighttpd**, **juise**, or **all**. When you specify the trace options, the command overwrites any previous trace option settings.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35
- **rest on page 43**
- Understanding the REST API on page 13

- Example: Using the REST API Explorer on page 20

Example: Configuring the REST API

Supported Platforms M Series, MX Series, PTX Series, T Series

This example demonstrates how to configure the REST API on a Junos OS device.

Requirements

- A routing, switching, or security device running Junos OS Release 14.2 or later is required.

Overview

This example configures the REST API on a Juniper Networks M10i Multiservice Edge Router. The example configures both HTTP and HTTPS access, with both lighttpd and juise tracing.

Configuration

CLI Quick Configuration

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

```
set system services rest control allowed-sources [192.0.2.0 198.51.100.0]
set system services rest control connection-limit 100
set system services rest http port 3000
set system services rest http addresses [203.0.113.0 203.0.113.1]
set system services rest https port 3443
set system services rest https addresses [203.0.113.2 203.0.113.3]
set system services rest https server-certificate testcert
set system services rest https cipher-list rsa-with-3des-ede-cbc-sha
set system services rest https mutual-authentication certificate-authority testca
set system services rest traceoptions flag all
set system services rest enable-explorer
```

Configuring the REST API

Step-by-Step Procedure

To configure the REST API:

1. Specify allowed IP addresses for incoming HTTP and HTTPS connections.

```
[edit]
user@R1# set system services rest control allowed-sources [192.0.2.0 198.51.100.0]
```

2. Specify the maximum number of allowed connections over both HTTP and HTTPS.

```
[edit]
```

```
user@R1# set system services rest control connection-limit 100
```

3. Set the TCP port for incoming HTTP connections.

```
[edit]
user@R1# set system services rest http port 3000
```

4. Set the addresses on which the server listens for incoming HTTP connections.

```
[edit]
user@R1# set system services rest http addresses [203.0.113.0 203.0.113.1]
```

5. Set the TCP port for incoming HTTPS connections.

```
[edit]
user@R1# set system services rest https port 3443
```

6. Set the addresses on which the server listens for incoming HTTPS connections.

```
[edit]
user@R1# set system services rest https addresses [203.0.113.2 203.0.113.3]
```

7. Set the server certificate.

```
[edit]
user@R1# set system services rest https server-certificate testcert
```

8. Configure the set of ciphers the server can use to perform encryption and decryption functions.

```
[edit]
user@R1# set system services rest https cipher-list rsa-with-3des-ede-cbc-sha
```

9. (Optional) Set up TLS mutual authentication on both the server and client with a certificate.

```
[edit]
user@R1# set system services rest https mutual-authentication certificate-authority
  testca
```

10. (Optional) Configure trace options for lighttpd, juise, or both.

```
[edit]
user@R1# set system services rest traceoptions flag all
```

11. (Optional) Enable the REST API Explorer.

```
[edit]
user@R1# set system services rest enable-explorer
```

Results

```
system {
   services {
      rest {
         control {
            allowed-sources [ 192.0.2.0 198.51.100.0 ];
            connection-limit 100;
         }
         enable-explorer;
         http {
            addresses [ 203.0.113.0 203.0.113.1 ];
            port 3000;
         }
         https {
            port 3443;
            addresses [ 203.0.113.2 203.0.113.3 ];
            server-certificate testcert;
            cipher-list rsa-with-3des-ede-cbc-sha;
            mutual-authentication {
                certificate-authority testca;
            }
         }
         traceoptions {
            flag all;
         }
      }
   }
}
```

Verification

Verifying REST API Configuration

Purpose Confirm that the REST API configuration is working properly on the device.

Action Display the REST API configuration by issuing the **show configuration system services rest** operational mode command.

```
user@R1> show configuration system services rest
http {
    port 3000;
    addresses [ 203.0.113.0 203.0.113.1 ];
}
https {
    port 3443;
    addresses [ 203.0.113.2 203.0.113.3 ];
    server-certificate testcert;
    cipher-list rsa-with-3des-ede-cbc-sha;
    mutual-authentication {
        certificate-authority testca;
    }
}
control {
    allowed-sources [ 192.0.2.0 198.51.100.0 ];
    connection-limit 100;
}
traceoptions {
    flag all;
}
enable-explorer;
```

Meaning This example configured both HTTP and HTTPS access on a Juniper Networks M10i Multiservice Edge Router. For HTTP access, the device listens on port 3000 and permits traffic from IP addresses 192.0.2.0, 198.51.100.0, 203.0.113.0, and 203.0.113.1. For a more secure connection, HTTPS access was configured with mutual authentication, using port

3443 and allowed IP addresses of 192.0.2.0, 198.51.100.0, 203.0.113.2, and 203.0.113.3. A connection limit of 100 has been configured for both HTTP and HTTPS, and both juise and lighttpd tracing has been enabled. By default, the REST API Explorer is disabled (see "Example: Using the REST API Explorer" on page 20).

Related Documentation

- Understanding the REST API on page 13
- Configuring the REST API on page 15
- Example: Using the REST API Explorer on page 20

Example: Using the REST API Explorer

Supported Platforms M Series, MX Series, PTX Series, T Series

This example demonstrates how to optionally use the REST API Explorer on a Junos OS device on which the REST API has been configured.

- Requirements on page 20
- Overview on page 20
- Configuration on page 20

Requirements

- An M Series, MX Series, T Series, or PTX Series device running Junos OS Release 14.2 or later is required.

Overview

The REST API Explorer allows you to conveniently test out single or multiple RPC calls. Its GUI provides you with options to select the HTTP method (GET or POST), the required output format (XML, JSON, or plain text), the RPC URL, the input data type when using POST requests (XML or plain text), and an exit-on-error condition. When you submit the request, the REST API Explorer displays the request header, response header, response body, and equivalent cURL request, all of which are useful to your development efforts.

Configuration

To use the REST API Explorer on any device on which the REST API has been configured, perform these tasks:

- Enabling the REST API Explorer on page 21
- Opening the REST API Explorer on page 22
- Executing a Single RPC Using an HTTP GET Request on page 22
- Executing a Single RPC Using an HTTP POST Request on page 23
- Executing Multiple RPCs on page 26
- Viewing Error Messages on page 27

Enabling the REST API Explorer

Step-by-Step Procedure

To enable the REST API Explorer:

1. Configure the REST API on the device.

   See “Configuring the REST API” on page 15 and “Example: Configuring the REST API” on page 17 for information and examples.

2. Check whether the REST API Explorer is enabled.

   Use the **show** command to see if **enable-explorer;** appears in the REST API configuration. If it appears, the REST API Explorer has been enabled. If it does not appear, you must enable the REST API Explorer.

   ```
   [edit]
   user@R1# show system services rest
     http;
     traceoptions {
       flag all;
     }
     enable-explorer;
   ```

3. Enable the REST API Explorer if necessary.

   Use the **set** command to ensure that **enable-explorer;** appears in the REST API configuration.

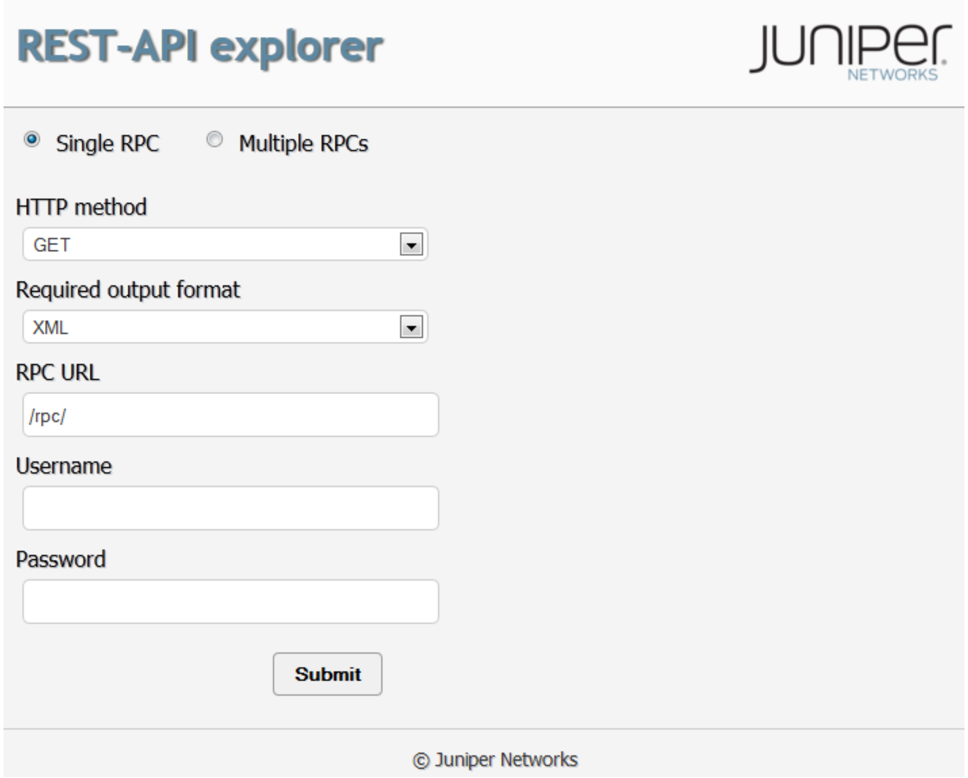
   ```
   [edit]
   user@R1# set system services rest enable-explorer
   ```

Opening the REST API Explorer

Step-by-Step Procedure

To open the REST API Explorer:

- Ensure that the REST API Explorer is enabled, open a browser, and go to the following URL: scheme://device-name:port (for example, https://mydevice:3000).



REST-API explorer

● Single RPC ○ Multiple RPCs

HTTP method: GET

Required output format: XML

RPC URL: /rpc/

Username

Password

Executing a Single RPC Using an HTTP GET Request

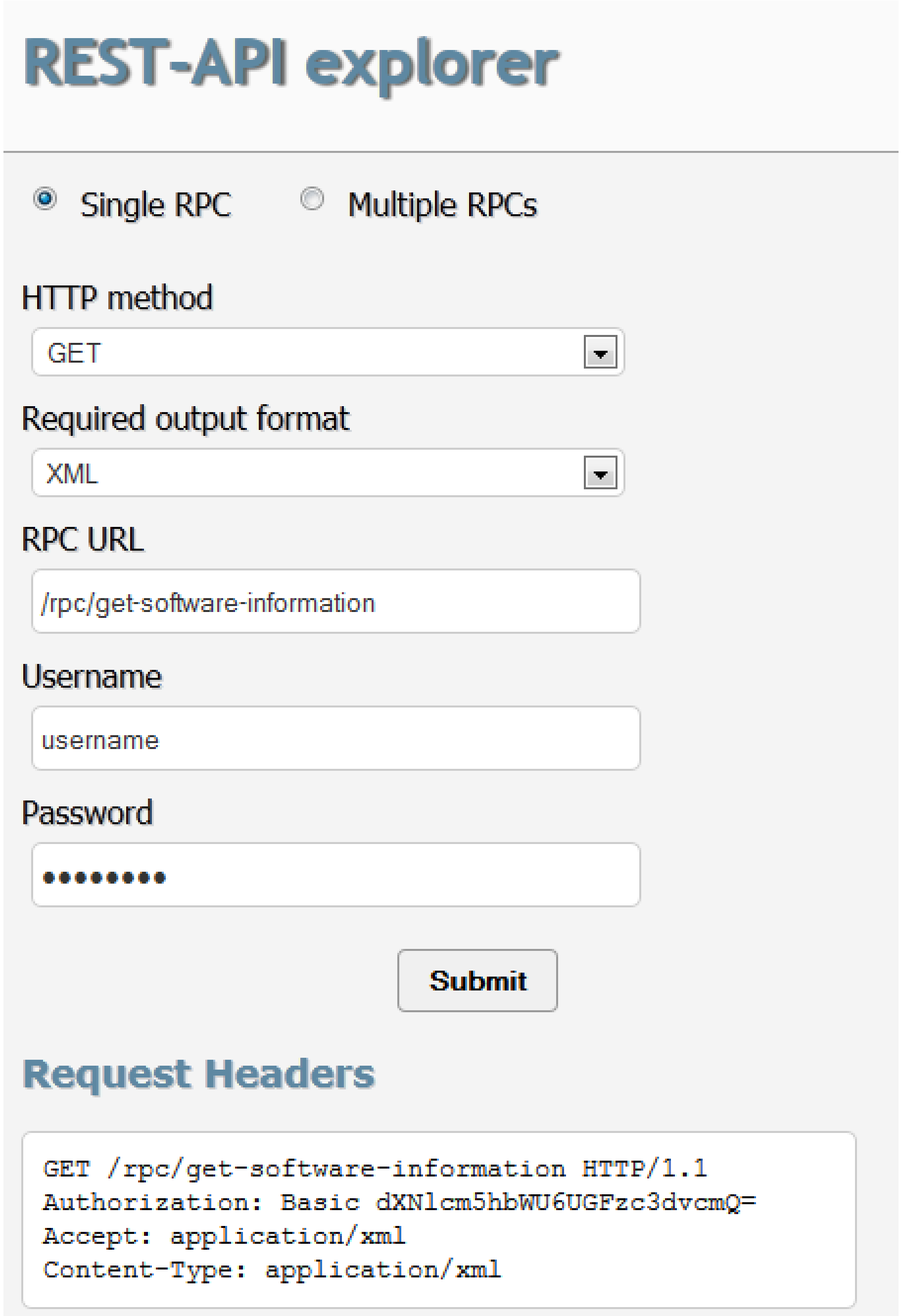
Step-by-Step Procedure

To execute a single RPC using an HTTP GET Request:

1. In the **HTTP method** drop-down list, select **GET**.
2. Enter the RPC URL endpoint.

   For example, type **/rpc/get-software-information**.
3. Enter your username and password.
4. Click **Submit**.

   In this example, the default output format, XML, is returned in the Response Body:



REST-API explorer

● Single RPC ○ Multiple RPCs

HTTP method

GET

Required output format

XML

RPC URL

/rpc/get-software-information

Username

username

Password

••••••••

Request Headers

```
GET /rpc/get-software-information HTTP/1.1
Authorization: Basic dXNlcm5hbWU6UGFzc3dvcmQ=
Accept: application/xml
Content-Type: application/xml
```

Executing a Single RPC Using an HTTP POST Request

Step-by-Step Procedure

To execute a single RPC using an HTTP POST Request:

1. In the **HTTP method** drop-down list, select **POST**.
2. In the **Required output format** drop-down list, select **JSON**.
3. Enter this RPC URL endpoint: **/rpc/get-software-information**.

4. Enter your username and password.

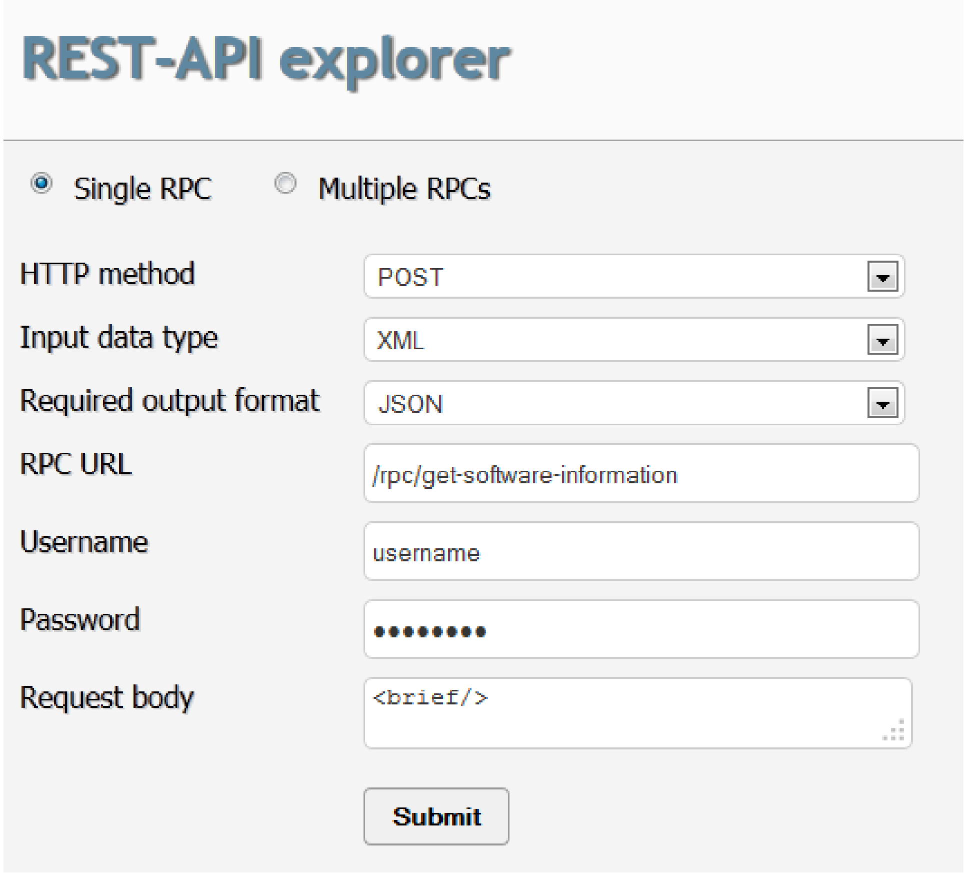
5. Enter the XML-formatted request in the **Request body** text area.

   For example:

   ```
   <brief/>
   ```

6. Click **Submit**.

   In this example, the JSON output format is returned in the Response Body:



REST-API explorer

● Single RPC ○ Multiple RPCs

HTTP method: POST

Input data type: XML

Required output format: JSON

RPC URL: /rpc/get-software-information

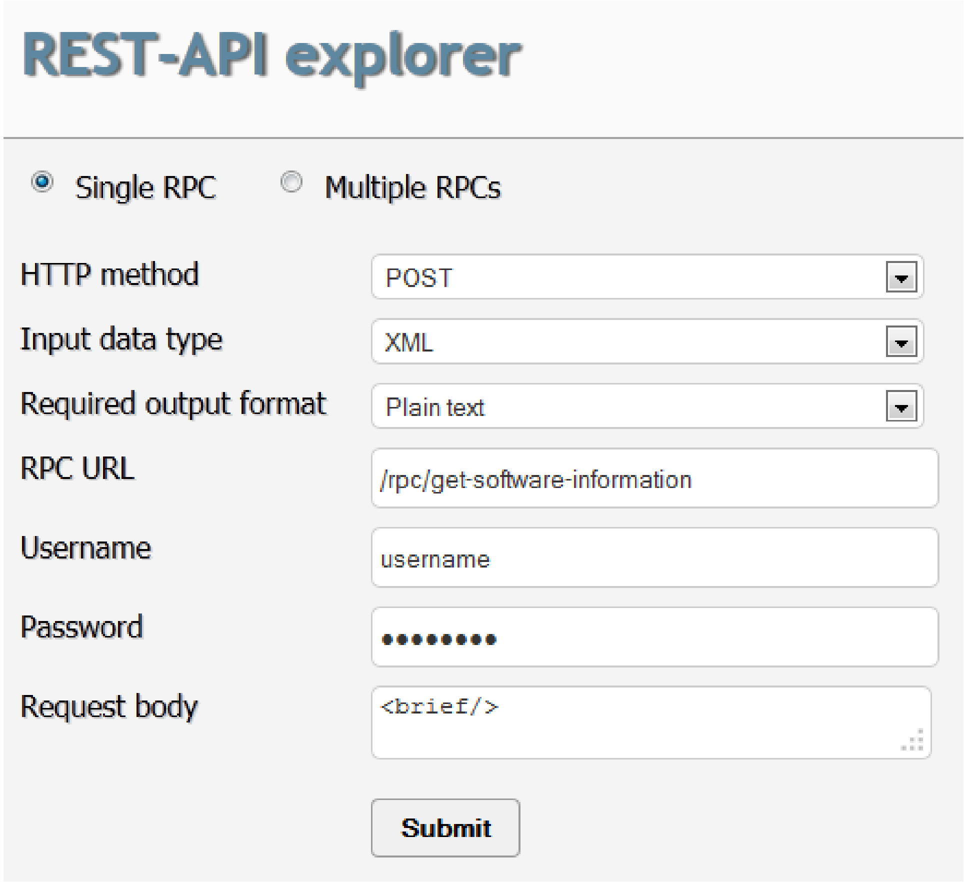
Username: username

Password: ●●●●●●●●●

Request body: `<brief/>`

7. If you prefer a different output format, select one of the available choices in the **Required output format** drop-down list.

   For example, you could select **Plain text**. When you click **Submit**, you will see plain text in the Response Body:



REST-API explorer

● Single RPC ○ Multiple RPCs

HTTP method: POST

Input data type: XML

Required output format: Plain text

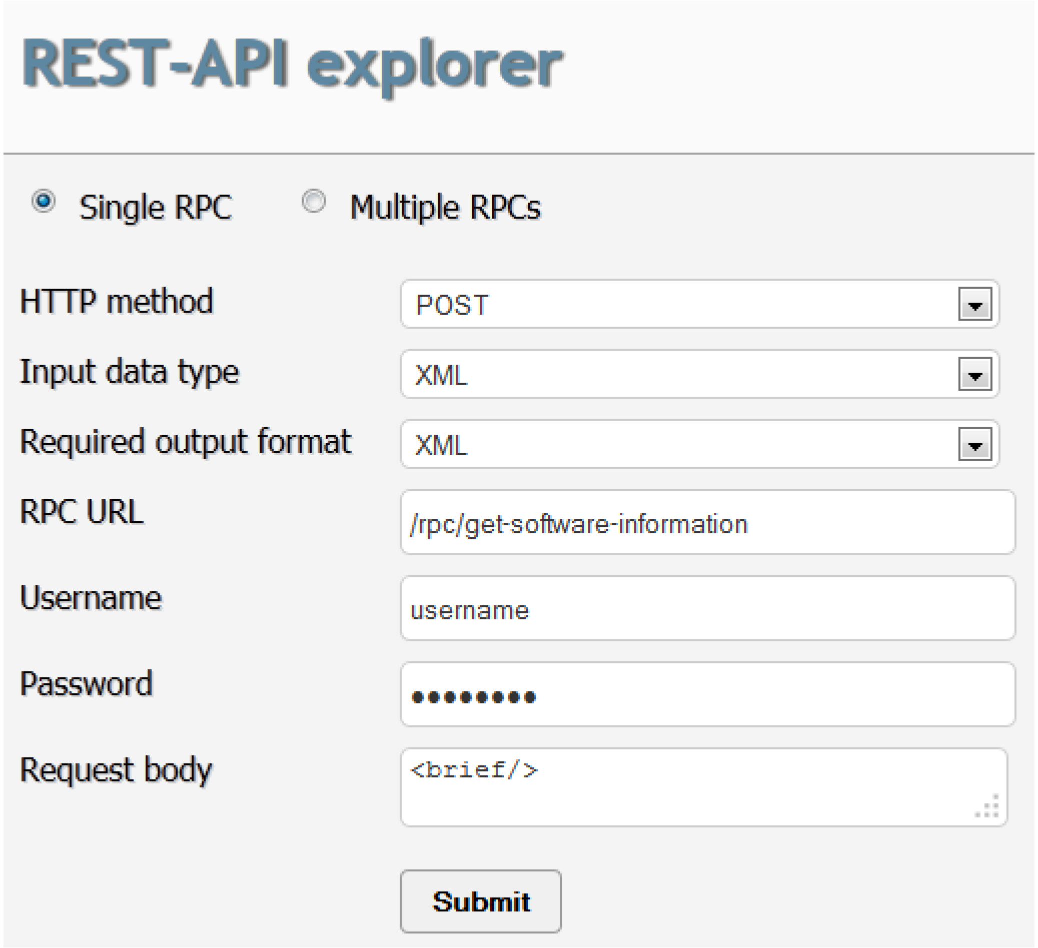
RPC URL: /rpc/get-software-information

Username: username

Password: ••••••••

Request body: `<brief/>`

Similarly, if you select **XML** in the **Required output format** drop-down list, the response body will contain XML-formatted information:



● Single RPC ○ Multiple RPCs

HTTP method: POST

Input data type: XML

Required output format: XML

RPC URL: /rpc/get-software-information

Username: username

Password: ••••••••

Request body: <brief/>

Executing Multiple RPCs

Step-by-Step Procedure

To execute multiple RPCs:

1. In the **HTTP method** drop-down list, select **POST**.

   This is always required when executing multiple RPCs.

2. To set a conditional exit in the event of an error, select the **Exit on error** checkbox.

3. Select an output format in the **Required output format** drop-down list.

   For example, you could select **JSON**.

4. This RPC URL endpoint will automatically populate: **/rpc?exit-on-error=1**.

5. Enter your username and password.

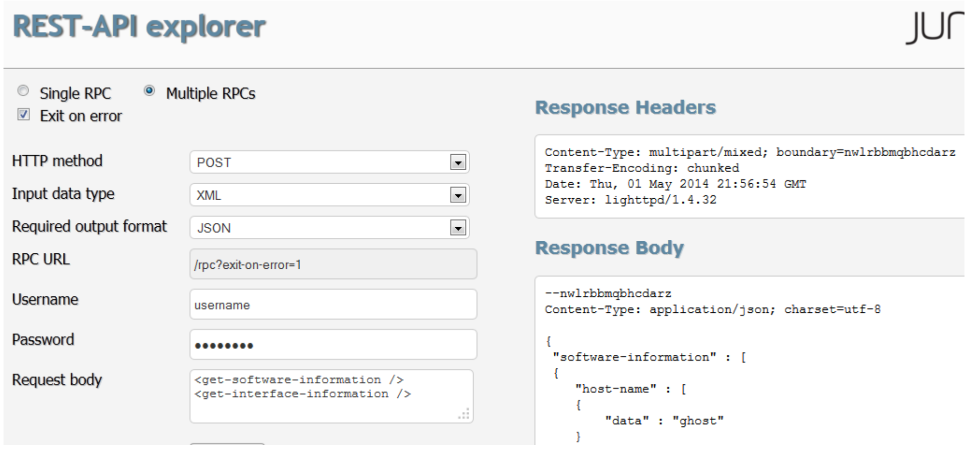
6. Enter the XML-formatted request in the **Request body** text area.

   For example:

```
<get-software-information />
<get-interface-information />
```

7. Click **Submit**.

   In this example, the JSON output format is returned in the Response Body:



REST-API explorer

○ Single RPC ● Multiple RPCs
☑ Exit on error

HTTP method: POST
Input data type: XML
Required output format: JSON
RPC URL: /rpc?exit-on-error=1
Username: username
Password: ••••••••
Request body:

```
<get-software-information />
<get-interface-information />
```

Response Headers

```
Content-Type: multipart/mixed; boundary=nwlrbbmqbhcdarz
Transfer-Encoding: chunked
Date: Thu, 01 May 2014 21:56:54 GMT
Server: lighttpd/1.4.32
```

Response Body

```
--nwlrbbmqbhcdarz
Content-Type: application/json; charset=utf-8

{
 "software-information" : [
 {
    "host-name" : [
    {
        "data" : "ghost"
    }
```

Viewing Error Messages

Step-by-Step Procedure

When executing multiple RPCs, an error might occur. If you select the **Exit on error** checkbox, an error message will appear in the output if an error occurs.

To view error messages:

1. In the **HTTP method** drop-down list, select **POST**.

   This is always required when executing multiple RPCs.

2. To set a conditional exit in the event of an error, select the **Exit on error** checkbox.

3. Select an output format in the **Required output format** drop-down list.

   For example, you could select **JSON**.

4. This RPC URL endpoint will automatically populate: **/rpc?exit-on-error=1**.

5. Enter your username and password.

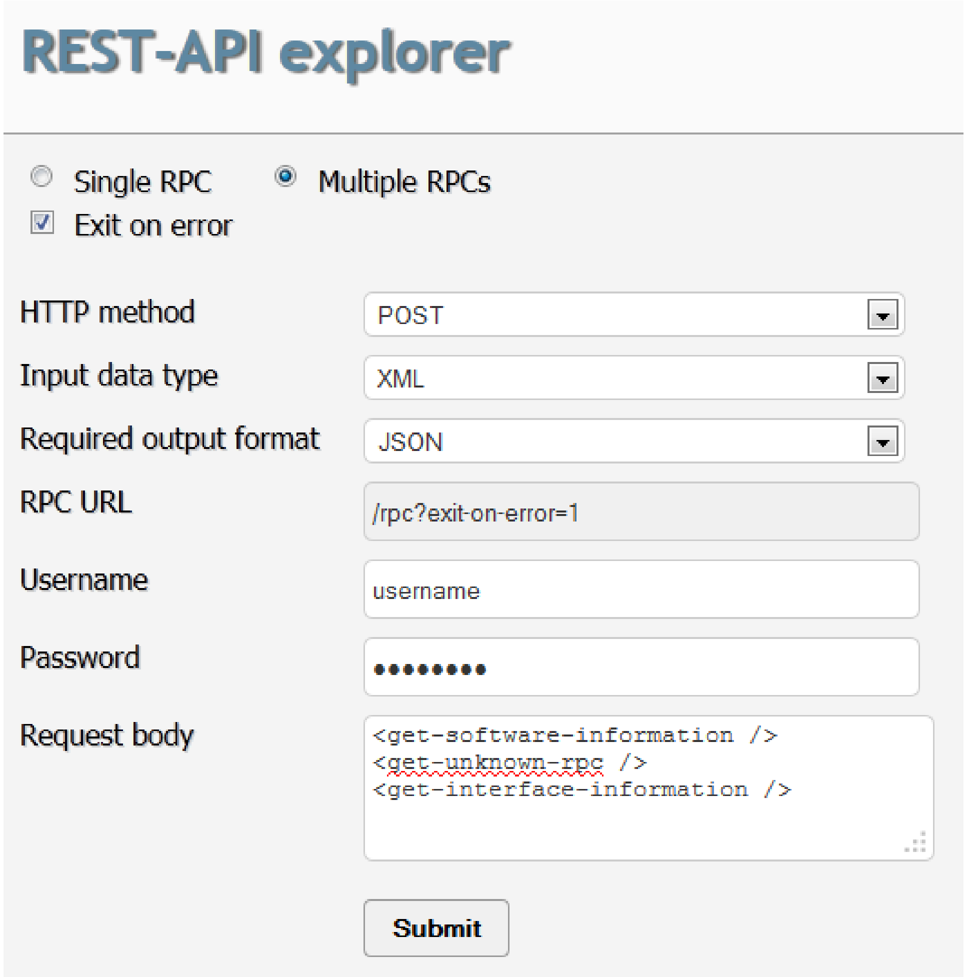
6. Enter the XML-formatted request containing an error in the **Request body** text area.

   For example:

```
<get-software-information />
<get-unknown-rpc />
<get-interface-information />
```

7. Click **Submit**.

   In this example, the JSON output format is returned in the Response Body, and you can see an XML-formatted error message at the end of the Response Body:



REST-API explorer

○ Single RPC ● Multiple RPCs

☑ Exit on error

HTTP method: POST

Input data type: XML

Required output format: JSON

RPC URL: /rpc?exit-on-error=1

Username: username

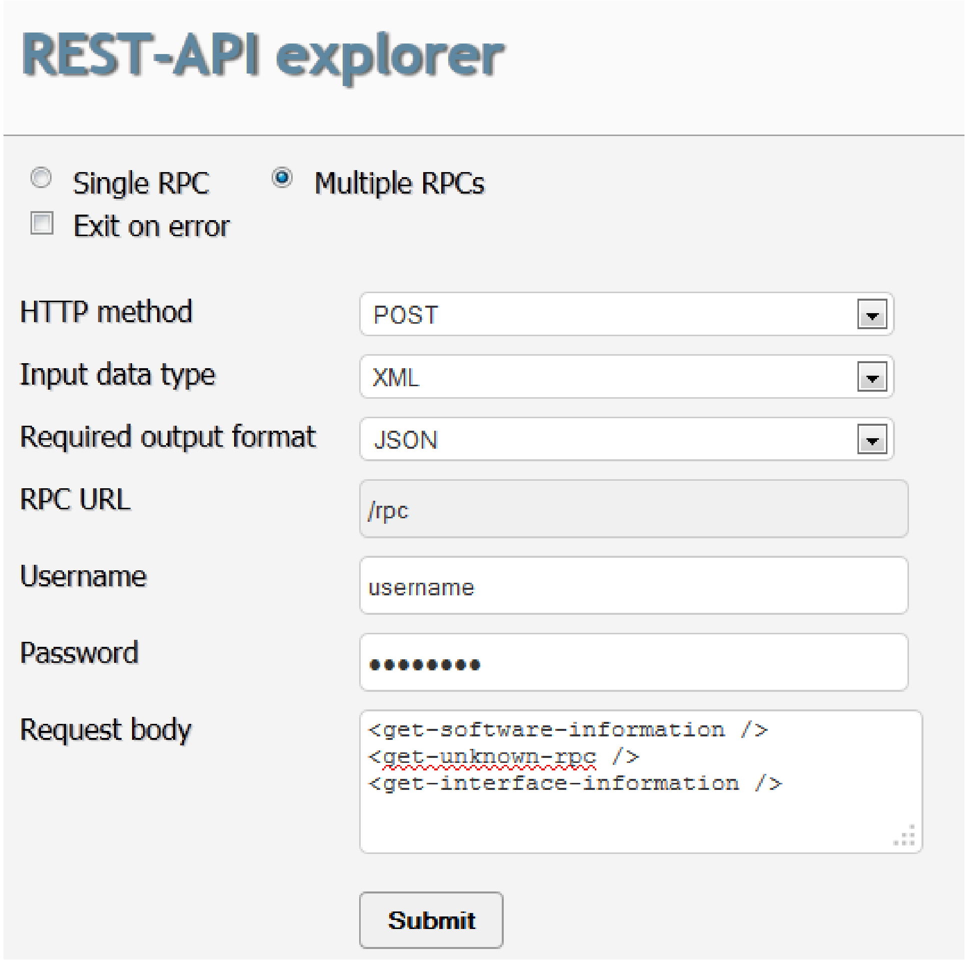
Password: ●●●●●●●●

Request body:

```
<get-software-information />
<get-unknown-rpc />
<get-interface-information />
```

8. If you do not select the **Exit on error** checkbox, an error message will appear in the Response Body if an error occurs.

   Execution will continue after the error is processed, and the results will also be included in the Response Body:



REST-API explorer

○ Single RPC ● Multiple RPCs

□ Exit on error

HTTP method	POST
Input data type	XML
Required output format	JSON
RPC URL	/rpc
Username	username
Password	●●●●●●●●

Request body

```
<get-software-information />
<get-unknown-rpc />
<get-interface-information />
```

Related Documentation

- Understanding the REST API on page 13
- Configuring the REST API on page 15

Submitting a GET Request to the REST API

Supported Platforms M Series, MX Series, PTX Series, T Series

For an **rpc** command, the general format of the endpoints is:

scheme://device-name:port/rpc/method[@attributes]/params

- **scheme**: **http** or **https**
- **method**: The name of any Junos OS **rpc** command. The **method** name is identical to the tag element. For more information, see the *Junos XML API Operational Developer Reference*.
- **params**: Optional parameter values (**name[=value]**).

To authenticate your request, submit the base64-encoded username and password included in the Authorization header:

```
curl -u "username:password" http://device-name:port/rpc/get-interface-information
```

To specify **rpc** data as a query string in the URI for GET requests, you can use a **?** following the URI with the **&** delimiter separating multiple arguments, or use the **/** delimiter, as shown in these equivalent cURL calls:

For example:

```
curl -u "username:password"
http://device-name:port/rpc/get-interface-information?interface-name=cbp0&snmp-index=1
curl -u "username:password"
http://device-name:port/rpc/get-interface-information/interface-name=cbp0/snmp-index=1
```

HTTP Accept headers can be used to specify the return format using one of the following Content-Type values:

- application/xml (the default)
- application/json
- text/plain
- text/html

For example, the following cURL call specifies an output format of JSON:

```
curl -u "username:password"
http://device-name:port/rpc/get-interface-information?interface-name=cbp0 –header
 "Accept: application/json"
```

You can also specify the output format using the optional **format** parameter.

For example, the **<get-software-information>** tag element retrieves software process revision levels. The following HTTPS GET request executes this command and retrieves the results in JSON format:

https://device-name:3000/rpc/get-software-information@format=json

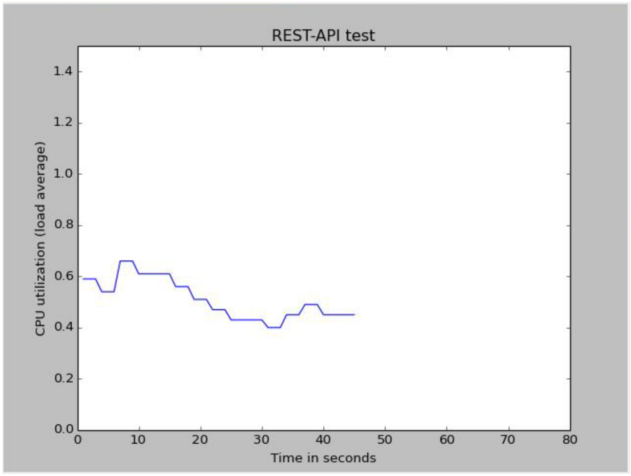
The following Python program uses the REST interface to execute the **get-route-engine-information** RPC, extracts the data from the response, and plots a graph of the CPU load average:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import requests

temp_y = 1
def update_line(num, data, line):
  if num == 0:
   return line,
  global temp_y
  x_data.append(num)
  if num is not 0 and num%8 == 1:
    r =
requests.get('scheme://device:port/rpc/get-route-engine-information@format=json',
 auth=('username', 'password'))
```

```
    if r: temp_y =
r.json()["route-engine-information"][0]["route-engine"][0]["load-average-one"][0]["data"]

  y_data.append(temp_y)
  line.set_data(x_data, y_data)
  return line,
fig1 = plt.figure()
x_data = []
y_data = []
l, = plt.plot([], [])
plt.xlim(0, 80)
plt.ylim(0, 1.5)
plt.xlabel('Time in seconds')
plt.ylabel('CPU utilization (load average)')
plt.title('REST-API test')
line_ani = animation.FuncAnimation(fig1, update_line, 80, fargs=(0, l), interval=1000,
blit=True)
plt.show()
```



Related Documentation

- Understanding the REST API on page 13
- Configuring the REST API on page 15
- Example: Using the REST API Explorer on page 20
- [edit system services rest] Hierarchy Level on page 35
- *| (pipe)*
- *Pipe (|) Filter Functions in the Junos OS Command-Line Interface*
- *Specifying the Output Format for Operational Information Requests in a NETCONF Session*

Submitting a POST Request to the REST API

Supported Platforms M Series, MX Series, PTX Series, T Series

Use an HTTP POST request to send single or multiple RPC requests to the REST API. You can use the POST request to do device configuration.

For a single **rpc** command, the general format of the endpoints is:

scheme://device-name:port/rpc/method[@attributes]/params

- **scheme**: **http** or **https**
- **method**: The name of any Junos OS **rpc** command. The **method** name is identical to the tag element. For more information, see the Junos XML Protocol Operations, Processing Instructions, and Response Tags in the *Junos XML Management Protocol Developer Guide* and the *Junos XML API Operational Developer Reference*.
- **params**: Optional parameter values (**name[=value]**).

To authenticate your request, submit the base64-encoded username and password included in the Authorization header:

```
curl -u "username:password" http://device-name:port/rpc/get-interface-information
```

To specify **rpc** data as a query string in the URI for POST requests, submit the query data in the POST body. In such cases you can specify the **Content-Type** as **text/plain** or **application/xml**, as shown in these equivalent cURL calls:

```
curl -u "username:password" http://device-name:port/rpc/get-interface-information
--header "Content-Type: text/plain" –d "interface-name=cbp0"
curl -u "username:password" http://device-name:port/rpc/get-interface-information
--header "Content-Type: application/xml" –d
"<interface-name>cbp0</interface-name>"
```

For both single and multiple RPC commands, HTTP Accept headers can be used to specify the return format using one of the following Content-Type values:

- application/xml (the default)
- application/json
- text/plain
- text/html

For example, the following cURL call specifies an output format of JSON:

```
curl -u "username:password" http://device-name:port/rpc -d <get-software-information
 /> –header "Accept: application/json"
```

You can also specify the output format using the optional **format** attribute:

```
curl -u "username:password" http://device-name:port/rpc -d "<get-software-information
 format=application/json'/>"
```

NOTE: The default Content-Type for POST requests containing arguments in the body is application/xml. If you want to use any other content, such as a query string, you can specify a Content-Type of text/plain. Specify the **format** attribute in configuration commands.

When executing multiple **rpc** commands in a single request, the general format of the endpoint is:

scheme://device-name:port/rpc

The RPCs must be provided as XML data in the POST body. The Content-Type for the response is multipart/mixed, with boundary and subtype associated with the output from each RPC execution. The format specified in the Accept header is used as the output format for each of the RPCs if they are missing a **format** attribute. If an Accept header is not specified and no **format** attribute is specified in a given RPC, the default output format is XML. For example, to send a single HTTP request to execute the RPCs **get-software-information** and **get-interface-information**, submit a POST request to **/rpc** with **"Auth: Basic <base64hash>"**, **"Content-Type: application/xml"**. The POST body would contain:

<get-software-information/> <get-interface-information/>

Here is a cURL call using this POST body:

```
curl -u "username:password" http://device-name:port/rpc -d
"<get-software-information/><get-interface-information/>"
```

The output from the request, containing XML as the default, would appear as follows:

```
HTTP/1.1 200 OK
Content-Type: multipart/mixed; boundary=fkj49sn38dcn3
Transfer-Encoding: chunked
Date: Thu, 20 Mar 2014 11:01:27 GMT
Server: lighttpd/1.4.32
--fkj49sn38dcn3
Content-Type: application/xml


<software-information>
<host-name>...</host-name>
...
</software-information>
--fkj49sn38dcn3
Content-Type: application/xml

<interface-information>
<physical-interface>...</physical-interface>
</interface-information>
--fkj49sn38dcn3--
```

You can also specify the output format for each of the elements in the POST body. For example, the following request emits JSON for the **get-interface-information** RPC and plain text for the **get-software-information** RPC:

```
curl -u "username:password" http://device-name:port/rpc
-d "<get-interface-information/><get-software-information format='text/plain'/>"
—header "Accept: application/json"
```

When executing multiple RPCs, if an error occurs, the default behavior is to ignore the error and continue execution. If you want to exit when the first error is encountered, specify the **stop-on-error** flag in the URI. For example, the following request configures the device and terminates if an error is encountered:

```
curl -u "username:password" http://device-name:port/rpc?stop-on-error=1
-d "<lock-configuration/>
 <load-configuration>
  <configuration><system><hostname>foo</hostname></system></configuration>

 </load-configuration>
 <commit/>
  <unlock-configuration/>"
```

Related Documentation

- Understanding the REST API on page 13
- [edit system services rest] Hierarchy Level on page 35
- | *(pipe)*
- *Pipe (|) Filter Functions in the Junos OS Command-Line Interface*
- *Specifying the Output Format for Operational Information Requests in a NETCONF Session*
- Configuring the REST API on page 15
- Example: Using the REST API Explorer on page 20

CHAPTER 3

Configuration Statements

[edit system services rest] Hierarchy Level

Supported Platforms M Series, MX Series, PTX Series, T Series

```
system {
  services {
    rest {
      control {
        allowed-sources [ value-list ];
        connection-limit limit;
      }
      enable-explorer;
      http {
        addresses [ address-list ];
        port port-number;
      }
      https {
        addresses [ address-list ];
        cipher-list [cipher-1 cipher-2 cipher-3 ... ];
```

```
                mutual-authentication {
                    certificate-authority certificate-authority-profile-name;
                }
                port port-number;
                server-certificate local-certificate-identifier;
            }
            traceoptions {
                flag flag;
            }
        }
    }
}
```

Related Documentation

- rest on page 43
- Understanding the REST API on page 13
- Configuring the REST API on page 15

addresses (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax addresses [*address-list*];

Hierarchy Level [edit system services rest http] ,
[edit system services rest https]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify IP addresses for incoming connections.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

allowed-sources (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
allowed-sources [ value-list ];
```

Hierarchy Level [edit system services rest control]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify the allowed source IP addresses for the REST API process.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

certificate-authority (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
certificate-authority certificate-authority-profile-name;
```

Hierarchy Level [edit system services rest https mutual-authentication]

Release Information Statement introduced in Junos OS Release 14.2.

Description Set the server certificate authority profile when configuring mutual authentication.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

cipher-list (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax cipher-list [*cipher-1 cipher-2 cipher-3* ...];

Hierarchy Level [edit system services rest https]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify the set of ciphers the server can use to perform encryption and decryption functions. If this option is not configured, the server accepts any supported suite that is available.

Options

- **rsa-with-RC4-128-md5**— RSA, 128- bit RC4, MD5 hash
- **rsa-with-RC4-128-sha**— RSA, 128-bit RC4, SHA hash
- **rsa-with-3DES-ede-cbc-sha**— RSA, 3DES EDE/CBC, SHA hash
- **dhe-rsa-with-3DES-ede-cbc-sha**— DHE/RSA, 3ES/EDE CBC, SHA hash
- **rsa-with-aes-128-cbc-sha**— RSA, 128-bit AES/CBC, SHA hash
- **dhe-rsa-with-aes-128-cbc-sha**— DHE/RSA, 128-bit AES/CBC, SHA hash
- **rsa-with-aes-256-cbc-sha**— RSA, 256 bit AES/CBC, SHA hash
- **dhe-rsa-with-aes-256-cbc-sha**— DHE/RSA, 256 bit AES/CBC, SHA hash
- **ecdhe-rsa-with-RC4-128-sha**— ECDHE/RSA, 128-bit RC4, SHA hash
- **ecdhe-rsa-with-3DES-ede-cbc-sha**— ECDHE/RSA, 128-bit 3DES EDE/CBC SHA hash
- **ecdhe-rsa-with-aes-128-cbc-sha**— ECDHE/RSA, 128-bit AES/CBC, SHA hash
- **ecdhe-rsa-with-aes-256-cbc-sha**— ECDHE/RSA, 256 bit AES/CBC, SHA hash
- **rsa-with-aes-128-cbc-SHA256**— RSA, 128-bit AES/CBC, SHA256 hash
- **rsa-with-aes-256-cbc-SHA256**— RSA, 256 bit AES/CBC, SHA256 hash
- **dhe-rsa-with-aes-128-cbc-SHA256**— DHE/RSA, 128-bit AES/CBC, SHA256 hash
- **dhe-rsa-with-aes-256-cbc-SHA256**— DHE/RSA, 256 bit AES/CBC, SHA256 hash
- **rsa-with-aes-128-gcm-SHA256**— RSA, 128-bit AES/GCM, SHA256 hash
- **rsa-with-aes-256-gcm-SHA384**— RSA, 256 bit AES/GCM, SHA384 hash
- **dhe-rsa-with-aes-128-gcm-SHA256**— DHE/RSA, 128-bit AES/GCM, SHA256 hash
- **dhe-rsa-with-aes-256-gcm-SHA384**— DHE/RSA, 256 bit AES/GCM, SHA384 hash
- **ecdhe-rsa-with-aes-128-cbc-SHA256**— ECDHE/RSA, 128-bit AES/CBC, SHA256 hash
- **ecdhe-rsa-with-aes-256-cbc-SHA384**— ECDHE/RSA, 256 bit AES/CBC, SHA384 hash
- **ecdhe-rsa-with-aes-128-gcm-SHA256**— ECDHE/RSA, 128-bit AES/GCM, SHA256 hash
- **ecdhe-rsa-with-aes-256-gcm-SHA384**— ECDHE/RSA, 256 bit AES/GCM, SHA384 hash

NOTE: For Junos OS in FIPS mode, only the following FIPS-compliant cipher algorithms are supported:

- rsa-with-aes-256-gcm-SHA384— RSA, 256 bit AES/GCM, SHA384 hash
- dhe-rsa-with-aes-128-gcm-SHA256— DHE/RSA, 128-bit AES/GCM, SHA256 hash
- dhe-rsa-with-aes-256-gcm-SHA384— DHE/RSA, 256 bit AES/GCM, SHA384 hash
- ecdhe-rsa-with-aes-128-gcm-SHA256— ECDHE/RSA, 128-bit AES/GCM, SHA256 hash
- ecdhe-rsa-with-aes-256-gcm-SHA384— ECDHE/RSA, 256 bit AES/GCM, SHA384 hash

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

connection-limit (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax connection-limit *limit*;

Hierarchy Level [edit system services rest control]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify the maximum number of simultaneous connections for the REST API process.

Options *limit*—Maximum number of simultaneous connections (IPv4 only).

Range: 1 through 1024

Default: 64

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

control (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
control {
  allowed-sources [ value-list ];
  connection-limit limit;
}
```

Hierarchy Level [edit system services rest]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify the allowed source IP addresses and maximum number of simultaneous connections for the REST API process.

The remaining statements are explained separately.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

enable-explorer (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
enable-explorer;
```

Hierarchy Level [edit system services rest]

Release Information Statement introduced in Junos OS Release 14.2.

Description Enable the REST API Explorer. This GUI is disabled by default, and can be enabled by specifying **set system services rest enable-explorer**. To disable the REST API Explorer, specify **delete system services rest enable-explorer**. To learn more about the REST API Explorer, see "Example: Using the REST API Explorer" on page 20.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35
- Example: Using the REST API Explorer on page 20

http (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
http {
  addresses [ address-list ];
  port port-number;
}
```

Hierarchy Level [edit system services rest]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify unencrypted HTTP connection settings, including addresses for incoming connections and the port number.

The remaining statements are explained separately.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

https (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
https {
  addresses [ address-list ];
  cipher-list [cipher-1 cipher-2 cipher-3 ... ];
  mutual-authentication {
    certificate-authority certificate-authority-profile-name;
  }
  port port-number;
  server-certificate local-certificate-identifier;
}
```

Hierarchy Level [edit system services rest]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify encrypted HTTPS connection settings, including addresses for incoming connections, the port number, preferred cipher suites, and server certificate.

The remaining statements are explained separately.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

mutual-authentication (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
mutual-authentication {
  certificate-authority certificate-authority-profile-name;
}
```

Hierarchy Level [edit system services rest https]

Release Information Statement introduced in Junos OS Release 14.2.

Description Enable SSL/TLS mutual authentication. The server certificate must be set when configuring mutual authentication.

The remaining statement is explained separately.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

port (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax port *port-number*;

Hierarchy Level [edit system services rest http],
[edit system services rest https]

Release Information Statement introduced in Junos OS Release 14.2.

Description Specify the port number.

Options *port-number*—Port number on which to accept HTTP or HTTPS connections.

Range: 1024 through 65535

Default: 3000 for HTTP, 3443 for HTTPS

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

rest

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
rest {
  control {
    allowed-sources [ value-list ];
    connection-limit limit;
  }
  enable-explorer;
  http {
    addresses [ address-list ];
    port port-number;
  }
  https {
    addresses [ address-list ];
    cipher-list [cipher-1 cipher-2 cipher-3 ... ];
    mutual-authentication {
      certificate-authority certificate-authority-profile-name;
    }
    port port-number;
    server-certificate local-certificate-identifier;
  }
  traceoptions {
    flag flag;
  }
}
```

Hierarchy Level [edit system services]

Release Information Statement introduced in Junos OS Release 14.2.

Description Execute Junos OS commands over HTTP or HTTPS using REST. Optionally, specify JSON output for operational and configuration commands.

The remaining statements are explained separately.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

server-certificate (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax server-certificate *local-certificate-identifier*;

Hierarchy Level [edit system services rest https]

Release Information Statement introduced in Junos OS Release 14.2.

Description Set the server certificate when configuring SSL/TLS mutual authentication.

Options *local-certificate-identifier*—The server certificate. This must be set when configuring SSL/TLS mutual authentication.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

traceoptions (REST API)

Supported Platforms M Series, MX Series, PTX Series, T Series

Syntax

```
traceoptions {
  flag flag;
}
```

Hierarchy Level [edit system services rest]

Release Information Statement introduced in Junos OS Release 14.2.

Description Define tracing operations for the REST API service.

Options flag *flag*—Tracing operation to perform. To specify more than one tracing operation, specify **all**. REST API tracing options include:

- **all**—All tracing operations. A combination of the juise and lighttpd tracing operations.
- **juise**—Trace juise operations. Trace information is captured in **/var/chroot/rest-api/log/juise**.
- **lighttpd**—Trace lighttpd operations. Trace information is captured in **/var/chroot/rest-api/log/lighttpd**.

Required Privilege Level system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

Related Documentation

- [edit system services rest] Hierarchy Level on page 35

CHAPTER 4

Index

Index