

Junos® OS

REST API Guide

Published
2025-06-24

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

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

Copyright © 2025 Juniper Networks, Inc. All rights reserved.

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

About This Guide | iv

1

Overview

Understanding the REST API | 2

2

Configuring and Using the REST API

Configuring the REST API | 5

Example: Configuring the REST API | 7

Requirements | 7

Overview | 7

Configuration | 7

Verification | 11

Example: Using the REST API Explorer | 12

Requirements | 13

Overview | 13

Configuration | 13

Submitting a GET Request to the REST API | 23

Submitting a POST Request to the REST API | 27

3

Configuration Statements and Operational Commands

Junos CLI Reference Overview | 32

About This Guide

The Junos OS REST API is a Representational State Transfer (REST) interface that enables you to securely connect to Junos OS devices, execute remote procedure calls, use a REST API Explorer graphical user interface 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).

1

CHAPTER

Overview

IN THIS CHAPTER

- [Understanding the REST API | 2](#)
-

Understanding the REST API

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.



NOTE:

The REST API incoming request payload size cannot exceed 1174KB.
Workaround: Chunk the incoming REST API requests into a smaller size.

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 23](#) and ["Submitting a POST Request to the REST API" on page 27](#) 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 12](#).

Change History Table

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

Release	Description
24.4R1 & 24.4R1-EVO	Starting in Junos OS Release 24.4R1 and Junos OS Evolved Release 24.4R1, we've deprecated the compact statement at the <code>[edit system export-format state-data json]</code> hierarchy level.
17.3R1	Starting in Junos OS Release 17.3R1, Junos OS supports emitting operational state data in compact JSON format. To emit the JSON data in compact format, configure the <code>json compact</code> statement at the <code>[edit system export-format state-data]</code> hierarchy level. Otherwise, the device emits the JSON data in non-compact format by default.

RELATED DOCUMENTATION

Example: Using the REST API Explorer 12
Configuring the REST API 5
Submitting a GET Request to the REST API 23
Submitting a POST Request to the REST API 27
/ (pipe)
Filtering Operational Command Output
Specifying the Output Format for Operational Information Requests in a NETCONF Session

2

CHAPTER

Configuring and Using the REST API

IN THIS CHAPTER

- [Configuring the REST API | 5](#)
 - [Example: Configuring the REST API | 7](#)
 - [Example: Using the REST API Explorer | 12](#)
 - [Submitting a GET Request to the REST API | 23](#)
 - [Submitting a POST Request to the REST API | 27](#)
-

Configuring the REST API

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 12](#).
- **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.



NOTE: Elevated CPU Usage by systemd on ACX7100 Series– On ACX7100 platforms running Junos EVO, the `systemd` process may show higher-than-expected CPU usage under normal conditions. This occurs even when certain services (e.g., REST, syslog) are disabled. There is no functional impact observed.

RELATED DOCUMENTATION

rest

[Understanding the REST API | 2](#)

[Example: Using the REST API Explorer | 12](#)

Example: Configuring the REST API

IN THIS SECTION

- [Requirements | 7](#)
- [Overview | 7](#)
- [Configuration | 7](#)
- [Verification | 11](#)

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 `juisse` tracing.

Configuration

IN THIS SECTION

- [CLI Quick Configuration | 8](#)
- [Configuring the REST API | 8](#)
- [Results | 10](#)

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-edc-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-edc-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
```

12. Commit the configuration.

```
[edit]
user@R1# commit and-quit
```

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-edc-sha;
        mutual-authentication {
          certificate-authority testca;
        }
      }
      traceoptions {
        flag all;
      }
    }
  }
}
```



NOTE: When traceoptions are enabled for the REST API, the logs are stored at: /var/chroot/rest-api/var/log/lighttpd.

This location applies to platforms such as ACX7100 running Junos OS. The trace logs can assist in troubleshooting REST API-related issues.

Verification

IN THIS SECTION

- [Verifying REST API Configuration | 11](#)

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-edc-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 juisse and lighttpd tracing has been enabled. By default, the REST API Explorer is disabled (see ["Example: Using the REST API Explorer" on page 12](#)).

RELATED DOCUMENTATION

[Understanding the REST API | 2](#)

[Configuring the REST API | 5](#)

[Example: Using the REST API Explorer | 12](#)

Example: Using the REST API Explorer

IN THIS SECTION

- [Requirements | 13](#)
- [Overview | 13](#)
- [Configuration | 13](#)

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

- 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.

In Junos, both GET and POST can be used to execute a single operational RPC (such as get-software-information). While both methods can retrieve the same information, POST is required when you need to execute multiple RPCs or include XML data in the request body (using the -d option).

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

IN THIS SECTION

- [Enabling the REST API Explorer | 14](#)
- [Opening the REST API Explorer | 14](#)
- [Executing a Single RPC Using an HTTP GET Request | 15](#)
- [Executing a Single RPC Using an HTTP POST Request | 16](#)
- [Executing Multiple RPCs | 19](#)
- [Viewing Error Messages | 20](#)

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

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 5](#) and ["Example: Configuring the REST API" on page 7](#) 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 JUNIPER NETWORKS

☒ Single RPC ☐ Multiple RPCs

HTTP method
GET

Required output format
XML

RPC URL
/rpc/

Username

Password

Submit

© Juniper Networks

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
••••••••

Submit

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:

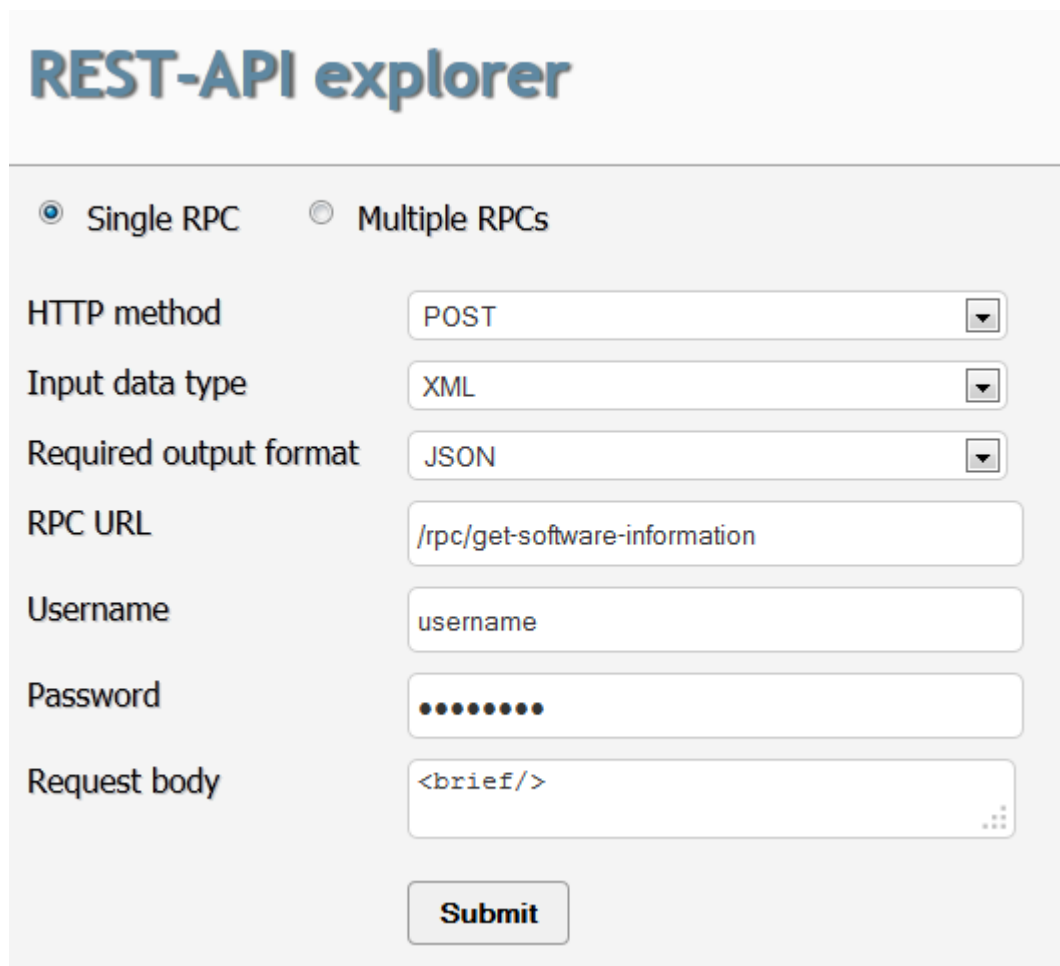
1. In the **HTTP method** drop-down list, select **POST**.
2. In the **Required output format** drop-down list, select **JSON**.
3. Enter this RPC URL endpoint: `/rpc/get-software-information`.
4. Enter your username and password.
5. Enter the XML-formatted request in the **Request body** text area.

For example:

```
<brief/>
```

6. Click **Submit**.

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



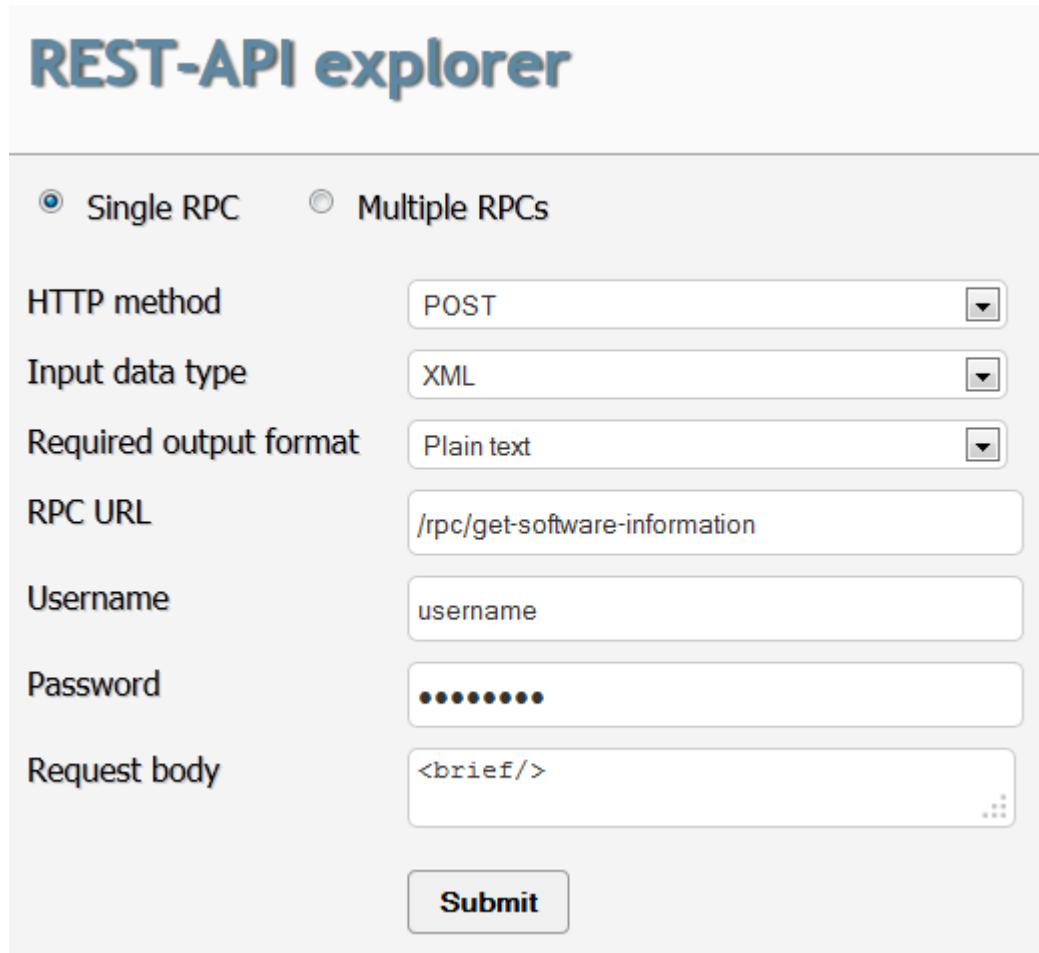
The screenshot shows the 'REST-API explorer' interface. At the top, there are two radio buttons: 'Single RPC' (selected) and 'Multiple RPCs'. Below this, there are several input fields and a 'Submit' button. The 'HTTP method' is set to 'POST'. The 'Input data type' is set to 'XML'. The 'Required output format' is set to 'JSON'. The 'RPC URL' is set to '/rpc/get-software-information'. The 'Username' is set to 'username'. The 'Password' is masked with dots. The 'Request body' is set to '<brief/>'. A 'Submit' button is at the bottom.

Field	Value
HTTP method	POST
Input data type	XML
Required output format	JSON
RPC URL	/rpc/get-software-information
Username	username
Password
Request body	<brief/>

Submit

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:



The screenshot shows the 'REST-API explorer' interface. At the top, there are two radio buttons: 'Single RPC' (selected) and 'Multiple RPCs'. Below this, there are several input fields and a submit button:

- HTTP method:** A dropdown menu with 'POST' selected.
- Input data type:** A dropdown menu with 'XML' selected.
- Required output format:** A dropdown menu with 'Plain text' selected.
- RPC URL:** A text input field containing '/rpc/get-software-information'.
- Username:** A text input field containing 'username'.
- Password:** A text input field with masked characters (dots).
- Request body:** A text input field containing '
'. There is a small icon in the bottom right corner of this field.
- Submit:** A button at the bottom of the form.

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

The screenshot shows the 'REST-API explorer' interface. At the top, there are two radio buttons: 'Single RPC' (selected) and 'Multiple RPCs'. Below this, there are several input fields and a submit button:

- HTTP method:** A dropdown menu with 'POST' selected.
- Input data type:** A dropdown menu with 'XML' selected.
- Required output format:** A dropdown menu with 'XML' selected.
- RPC URL:** A text input field containing '/rpc/get-software-information'.
- Username:** A text input field containing 'username'.
- Password:** A text input field with masked characters (dots).
- Request body:** A text input field containing '
brief/>'. There is a small icon in the bottom right corner of this field.
- Submit:** A button at the bottom of the form.

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:

The screenshot shows the REST-API explorer interface. On the left, the configuration panel has the following settings: **Single RPC** (unselected), **Multiple RPCs** (selected), **Exit on error** (checked), **HTTP method** (POST), **Input data type** (XML), **Required output format** (JSON), **RPC URL** (/rpc?exit-on-error=1), **Username** (username), **Password** (masked with dots), and **Request body** containing the XML request. On the right, the **Response Headers** section shows: Content-Type: multipart/mixed; boundary=nwlrbbmqbhdarz, Transfer-Encoding: chunked, Date: Thu, 01 May 2014 21:56:54 GMT, and Server: lighttpd/1.4.32. The **Response Body** section shows the JSON response: --nwlrbbmqbhdarz, Content-Type: application/json; charset=utf-8, and a JSON object with "software-information" containing an array of objects, one of which has "host-name" containing an array with "data" set to "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

Input data type

Required output format

RPC URL

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

Input data type

Required output format

RPC URL

Username

Password

Request body

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

RELATED DOCUMENTATION

[Understanding the REST API | 2](#)

[Configuring the REST API | 5](#)

Submitting a GET Request to the REST API

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 Explorer](#).
- `params`: Optional parameter values (`name[=value]`).

To authenticate your request, you can use one of the following methods. We recommend using the `netrc` option because it is more secure.

- Submit the base64-encoded username and password included in the Authorization header.

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

- Alternatively, use a `.netrc` file to store the credentials.
 1. In the user's home directory, create the `.netrc` file, and specify the hostname, username, and password for the remote device. For example:

```
user@host:~$ cat ~/.netrc
machine 172.16.2.1
login username
password password
```

2. Ensure that only the user has read and write permission for the file.

```
user@host:~$ chmod 600 .netrc
```

3. In the request, specify the `--netrc` option. For example:

```
user@host:~$ curl --netrc http://172.16.2.1:3000/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 --netrc http://device-name:port/rpc/get-interface-information?interface-name=cbp0&snmp-index=1
curl --netrc http://device-name:port/rpc/get-interface-information/interface-name=cbp0/snm-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 --netrc http://device-name:port/rpc/get-interface-information?interface-name=cbp0 --header
"Accept: application/json"
```

You can also specify the output format using the Junos OS RPC's 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:port/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. The `requests` module automatically checks the user's `.netrc` file for credentials associated with the specified device. Alternatively, you can include the `auth=(username, password)` argument in the request to supply credentials.

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

def update_line(num, data, line):
    if num == 0:
        return line,
    global temp_y
```

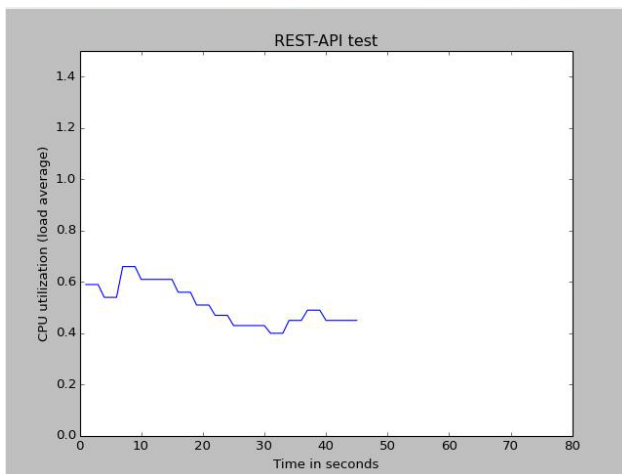
```

x_data.append(num)
if num != 0 and num%8 == 1:
    r = requests.get('http://' + device + '/rpc/get-route-engine-information@format=json')
    if r: temp_y = r.json()['route-engine-information'][0]['route-engine'][0]['load-average-
one'][0]['data']
    y_data.append(float(temp_y))
    line.set_data(x_data, y_data)
    return line,

device = input('Enter device:port ')

temp_y = 1
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, 1), interval=1000, blit=True)
plt.show()

```



RELATED DOCUMENTATION

[Understanding the REST API | 2](#)

[Configuring the REST API | 5](#)

*/ (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

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

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 API Explorer](#).
- params: Optional parameter values (name[=value]).

To authenticate your request, you can use one of the following methods. We recommend using the netrc option because it is more secure.

- Submit the base64-encoded username and password included in the Authorization header.

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

- Alternatively, use a **.netrc** file to store the credentials.
 1. In the user's home directory, create the **.netrc** file, and specify the hostname, username, and password for the remote device. For example:

```
user@host:~$ cat ~/.netrc
machine 172.16.2.1
login username
password password
```

2. Ensure that only the user has read and write permission for the file.

```
user@host:~$ chmod 600 .netrc
```

3. In the request, specify the `--netrc` option. For example:

```
user@host:~$ curl --netrc http://172.16.2.1:3000/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 --netrc http://device-name:port/rpc/get-interface-information --header "Content-Type: text/plain" -d "interface-name=cbp0"
curl --netrc 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 --netrc 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 --netrc http://device-name:port/rpc -d "<get-software-information format='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 you do not specify an Accept header or the format attribute 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 --netrc 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 --netrc http://device-name:port/rpc
-d "<get-interface-information/><get-software-information format='text' />"
--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 --netrc 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 | 2](#)

/ (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 | 5](#)

[Example: Using the REST API Explorer | 12](#)

3

CHAPTER

Configuration Statements and Operational Commands

IN THIS CHAPTER

- [Junos CLI Reference Overview | 32](#)
-

Junos CLI Reference Overview

We've consolidated all Junos CLI commands and configuration statements in one place. Read this guide to learn about the syntax and options that make up the statements and commands. Also understand the contexts in which you'll use these CLI elements in your network configurations and operations.

- [Junos CLI Reference](#)

Click the links to access Junos OS and Junos OS Evolved configuration statement and command summary topics.

- [Configuration Statements](#)
- [Operational Commands](#)