



# **DANOS-Vyatta edition**

## **Disaggregated Network Operating System Version 2009a**

**Remote Access API Configuration Guide**  
**October 2020**

# Contents

<b>Chapter 1. Copyright Statement</b> .....	<b>1</b>
<b>Chapter 2. Preface</b> .....	<b>2</b>
Document conventions.....	2
<b>Chapter 3. About This Guide</b> .....	<b>4</b>
<b>Chapter 4. Overview</b> .....	<b>5</b>
Introduction.....	5
Remote access API statements.....	5
API prerequisites.....	5
Enabling HTTPS on the router.....	6
Interacting with the API.....	6
Special-character handling.....	7
Response processing.....	8
Sample workflows.....	10
Deploying a multiple system configuration.....	10
Monitoring an interface across systems.....	10
<b>Chapter 5. Configuration Mode</b> .....	<b>11</b>
Overview.....	11
Summary of tasks.....	11
Session example.....	11
<b>Chapter 6. Configuration Mode Command Reference</b> .....	<b>18</b>
DELETE /rest/conf/<conf-id>.....	18
GET /rest/conf.....	19
GET /rest/conf/<conf-id>/<path>.....	21
POST /rest/conf.....	23
POST /rest/conf/<conf-id>/<cmd>.....	26
PUT rest/conf/<conf-id>/delete/<path>.....	27
PUT /rest/conf/<conf-id>/set/<path>.....	28
<b>Chapter 7. Operational Mode</b> .....	<b>31</b>
Overview.....	31
Summary of tasks.....	31

Example of one-time output.....	32
Example of continuous output.....	33
<b>Chapter 8. Operational Mode Command Reference.....</b>	<b>38</b>
DELETE /rest/op/<process-id>.....	38
GET /rest/op.....	39
GET /rest/op/<path>.....	41
GET/rest/op/<process-id>.....	42
POST /rest/conf.....	43
POST/rest/op/<path>.....	45
<b>Chapter 9. HTTP Status Codes.....</b>	<b>47</b>
<b>Chapter 10. List of Acronyms.....</b>	<b>48</b>

# Chapter 1. Copyright Statement

© 2020 IP Infusion Inc. All Rights Reserved.

This documentation is subject to change without notice. The software described in this document and this documentation are furnished under a license agreement or nondisclosure agreement. The software and documentation may be used or copied only in accordance with the terms of the applicable agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's internal use without the written permission of IP Infusion Inc.

IP Infusion Inc.  
3965 Freedom Circle, Suite 200  
Santa Clara, CA 95054  
+1 408-400-1900

<http://www.ipinfusion.com/>.

For support, questions, or comments via E-mail, contact:

[support@ipinfusion.com](mailto:support@ipinfusion.com).

Trademarks:

IP Infusion is a trademark of IP Infusion. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Use of certain software included in this equipment is subject to the IP Infusion, Inc. End User License Agreement at <http://www.ipinfusion.com/license>. By using the equipment, you accept the terms of the End User License Agreement.


# Chapter 2. Preface


## Document conventions


The document conventions describe text formatting conventions, command syntax conventions, and important notice formats used in this document.


### Notes, cautions, and warnings

Notes, cautions, and warning statements may be used in this document. They are listed in the order of increasing severity of potential hazards.

 **Note:** A Note provides a tip, guidance, or advice, emphasizes important information, or provides a reference to related information.

 **Attention:** An Attention statement indicates a stronger note, for example, to alert you when traffic might be interrupted or the device might reboot.

 **CAUTION:** A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

 **DANGER:** A Danger statement indicates conditions or situations that can be potentially lethal or extremely hazardous to you. Safety labels are also attached directly to products to warn of these conditions or situations.

### Text formatting conventions

Text formatting conventions such as boldface, italic, or Courier font are used to highlight specific words or phrases.

Format	Description
<b>bold text</b>	Identifies command names. Identifies keywords and operands.
<i>italic text</i>	Identifies emphasis. Identifies variables. Identifies document titles.
<code>Courier font</code>	Identifies CLI output. Identifies command syntax examples.

### Command syntax conventions

Bold and italic text identify command syntax components. Delimiters and operators define groupings of parameters and their logical relationships.

Convention	Description
<b>bold text</b>	Identifies command names, keywords, and command options.
<i>italic text</i>	Identifies a variable.
[ ]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ x   y   z }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options.
x   y	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, for example, passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, <i>member</i> [ <i>member</i> ...].
\	Indicates a "soft" line break in command examples. If a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

## **Chapter 3. About This Guide**

---

This guide describes how to use the Remote Access Application Programming Interface to remotely run commands on DANOS-Vyatta edition over HTTPs.

---

## Chapter 4. Overview

This chapter provides an overview of the router Remote Access API 2.0.

---

### Introduction

The router Remote Access API 2.0 enables remote command execution on a router over HTTPS by using a simple, coherent, and stateless interface. In addition to accessing the standard set of operational and configuration commands, the API provides process control and management features.

The API adheres to Representational State Transfer (REST) principles where possible, and uses the JavaScript Object Notation (JSON) format for data representation. Version 2.0 of the API largely replaces the original XML-based router Remote Access API 1.0.

The API interacts with the router through an operational mode/configuration mode model. The API modes are closely analogous to the corresponding CLI modes but include additional capability to support session and process management.

Commands that return response bodies provide JSON-formatted output, unless otherwise noted.


---

### Remote access API statements

REST statements are submitted through `HTTP GET`, `DELETE`, `PUT`, and `POST` commands by using the format of *HTTP-command REST-statement*, as in the following example:

```
GET /rest/conf
```

The specific result of any REST statement depends on the `HTTP` command that is used to submit the statement to the remote router.

 **Note:** The command reference material for REST statements in this guide includes the `HTTP` commands that are used to submit them; that is, the description uses the complete combination of `HTTP-command REST-statement` to describe how to interact with the remote router.

---

### API prerequisites

Using the router Remote Access API 2.0 requires the following:

- router that is running Release 6.0 or later software with HTTPS enabled
- Valid username and password on the router
- System to generate the HTTPS requests to the router



## Enabling HTTPS on the router

To enable HTTPS on the router, issue the following commands at the vyatta command prompt.

Description	Command
Enter configuration mode.	vyatta@vyatta:~\$ configure vyatta@vyatta#
Enable HTTPS on the system.	vyatta@vyatta# set service https
Commit the change.	vyatta@vyatta# commit
Save the change so that it is available after a system reboot.	vyatta@vyatta# save Saving configuration to '/config/config.boot'... Done
Return to operational mode.	vyatta@vyatta# exit vyatta@vyatta:~\$

## Interacting with the API

In general, an interaction with the router consists of an HTTPS request to the system that is followed by a response from the system. Each request includes in its HTTP header a command, the address of the remote system, the format of the response body that is expected, the specification version number, and a base64 encoding of a valid username-and-password pair (conforming to the basic access authorization originally defined in RFC 1945); these credentials are validated on the remote system's authorization system.

The following sample interaction shows a simple request and response.

Description	Command and Response
Clear the counters on dp0p1p1 on 10.0.0.232. The dnldHRhOnZ5YXR0YQ== character string is the base64 encoding of the vyatta:vyatta username-and-password pair.	POST /rest/op/clear/interfaces/dataplane/dp0p1p1/counters Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnldHRhOnZ5YXR0YQ==
System response.	HTTP/1.1 201 Created Transfer-Encoding: chunked Content-Type: application/json Location: rest/op/3479DEF17C6AF4D1 Date: Fri, 19 Feb 2010 21:27:57 GMT Server: lighttpd/1.4.19

For testing purposes, a simple way to create a request is to use a command such as `curl`, although any tool that allows you to construct an HTTPS request (such as `wget`, a browser, `python`, and so on) suffices.

For example, to generate the request that is shown in the previous example by using the `curl` command, you enter the command that is shown in the following example.

Description	Command
Use curl to generate an operational mode request, clear interfaces dataplane dp0p1p1 counters, by using the vyatta:vyatta username-and-password pair on the router at 10.0.0.232.	<pre>vyatta@R1:~\$curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X POST https://10.0.0.232/rest/op/clear/interfaces/dataplane/dp0p1p1/counters</pre>
System response.	<pre>HTTP/1.1 201 Created Transfer-Encoding: chunked Content-Type: application/json Location: rest/op/D462E45FCEFB5A5 Date: Fri, 19 Feb 2010 21:28:39 GMT Server: lighttpd/1.4.19</pre>

For examples of Perl scripts that use the API, refer to the following files on the router:

```
/opt/vyatta/sbin/vyatta-webgui2-shell.pl
/opt/vyatta/sbin/vyatta-webgui2-op.pl
```

## Special-character handling

Special characters that must be included within the command line of the request header must be formatted to be URL safe. For example, to set the address of dp0p1p1 to 10.0.0.231/24 by using the CLI, enter the following command in configuration mode:

```
set interfaces dataplane dp0p1p1 address 10.0.0.231/24
```

Within the API, however, the last slash character (/) is improperly parsed by the HTTP protocol, so it must be replaced by a sequence of escape characters to represent it. The escape encoding for the slash character is %2F; therefore, the URL-safe character string must be represented as follows:

```
rest/conf/set/interfaces/dataplane/dp0p1p1/address/10.0.0.231%2F24
```

The following table shows the encodings that are required for the various special characters.

**Table 1. Special-character encodings**

Special Character	Encoding
Tab	%09
Space	%20
"	%22

**Table 1. Special-character encodings (continued)**

Special Character	Encoding
#	%23
%	%25
&	%26
(	%28
)	%29
+	%2B
,	%2C
.	%2E
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
\	%5C
]	%5D
^	%5E
'	%60
{	%7B
	%7C
}	%7D
~	%7E

## Response processing

The response header contains the HTTP status code.

Before processing the response body, you should check that the request is successful by ensuring that the HTTP status code is a 200-level (that is, 2xxOK) response code. The Content-Type information returned in the response header should also be checked to verify that it is correct.

Response bodies of three types can be returned in response to a request:

- Empty body
- JSON-formatted body
- Plain text body

The body of the response is returned and its format depends on the requested HTTP Content-Type and the command that is issued through the API.

- Responses to requests that initiate actions (for example, to initiate the `show version` command) typically have empty bodies.
- Responses to requests that retrieve data (for example, a list of active operational processes) have JSON-formatted bodies.
- Responses to requests that contain operational mode command output (for example, output from the `show version` command) have plain text bodies.

JSON responses vary with the command that is issued. Additional command status codes can also be returned in the body of the response. [Response processing](#) shows a JSON-formatted response body—in this case, the currently active operational mode processes.

## A JSON-formatted response body

```
{
  "process": [
    {
      "username": "vyatta",
      "started": "1273252231",
      "updated": "8",
      "id": "02B3479CA1522F2A",
      "command": "ping 10.3.0.1"
    },
    {
      "username": "vyatta",
      "started": "1273262318",
      "updated": "8",
      "id": "A86BFCB1BC5E353E",
      "command": "show version"
    }
  ]
}
```

JSON responses follow standard JavaScript object construction where elements are hash entries, arrays, or nested combinations of entries and arrays. The response body that is shown previously has the following characteristics.

- There is a single hash: “process.” It points to an array of hashes.
- Each element of the array represents a distinct operational mode process (in this example, there are two of them) and contains information about that process.

- For each process, the response shows the user who initiated the command, time the command was initiated, number of updates that have been retrieved, ID of the process, and command that was executed (or is currently executing).

Processing of JSON-formatted responses is well supported by using support libraries that are available in most programming languages. Further information is located at <http://www.json.org>.

---

## Sample workflows

You can perform most tasks on the router through the API. The examples in this section show the work flow for some common or useful operations by using the API with a scripting or compiled language of choice.

### Deploying a multiple system configuration

1. Create new configuration sessions across the target system.
2. For each command issued by the source (user, script, or compiled language), dispatch a configuration mode `PUT` command to each target.
3. For each target, apply a commit action.
4. Verify the configuration by retrieving it through a configuration mode `GET` command.

### Monitoring an interface across systems

1. For each target, run the `show interfaces` command by using an operational mode `PUT` command.
2. For each target, poll on the process ID until the command is completed.
3. Sleep and continue to loop.

---

## Chapter 5. Configuration Mode


This chapter describes the functionality that is available in the configuration mode of the router Remote Access API 2.0.

---

### Overview

The configuration mode of the router Remote Access API allows you to update the system configuration of the remote router. The workflow for configuration by using the API is the same as that for using the system directly: begin a configuration session, make configuration changes, commit changes, optionally view them, and save them. Configuration mode is accessed by using the **rest/conf** URI prefix.

In configuration mode within the API, each configuration session creates a unique session ID that is used to reference the session as the URI **rest/conf/conf-id**. This ID must be explicitly removed when the session is completed to release the resources that are associated with the session.

 **Note:** Restarting the HTTPS service invalidates operational session IDs, but does not invalidate configuration session IDs.

---

### Summary of tasks

You can perform the following tasks by using the commands that are described in this chapter.

<a href="#">POST/rest/conf</a>	Start a configuration session and enter configuration mode.
<a href="#">PUT /rest/conf/&lt;conf-id&gt;/set/&lt;path&gt;</a>	Create or modify configuration information.
<a href="#">PUT rest/conf/&lt;conf-id&gt;/delete/&lt;path&gt;</a>	Delete configuration information.
<a href="#">POST /rest/conf/&lt;conf-id&gt;/&lt;cmd&gt;</a>	Enter a configuration action command, such as <code>commit</code> , <code>save</code> , <code>load</code> , <code>merge</code> , or <code>show</code> .
<a href="#">GET /rest/conf/&lt;conf-id&gt;/&lt;path&gt;</a>	Display configuration information.
<a href="#">GET /rest/conf</a>	List active configuration mode sessions.
<a href="#">DELETE /rest/conf/&lt;conf-id&gt;</a>	Exit the specified configuration session.

---

### Session example

The following example of a Remote Access API request to a router at 10.0.0.232 shows how to initiate a configuration mode session.

**Table 2. Requesting a configuration mode session**

Description	Command and Response
Request a new configuration mode session. This request is equivalent to the <code>configure</code> command in the CLI.	<pre>POST /rest/conf Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==</pre>

The HTTPS server that is located on port 443 receives this request and identifies it as a REST request to the configuration tree (**/rest/conf**). The MIME base64-encoded username:password (in this case, `dnlhdHRhOnZ5YXR0YQ==` is the base64 encoding of `vyatta:vyatta`) is then validated. After it is validated, the request runs using the credentials of the specified user (in this case, `vyatta`).


At this point, the action and the remaining elements in the header determine the specific processing required for this request. For this request, the action is to `POST` to the root of the **rest/conf** tree. The path and action instruct the remote router to create a new configuration session for this request and return a unique ID for this configuration session.

The response that is generated by the remote system follows.

**Table 3. System response that is the request for a configuration mode session**

Description	Command and Response
System response.	<pre>HTTP/1.1 201 Created Transfer-Encoding: chunked Content-Type: application/json Location: rest/conf/2957FCA1FE3B3C Date: Wed, 05 May 2010 18:52:48 GMT Server: lighttpd/1.4.19</pre>

The 201 response code indicates that the request was successful and that the remote system was able to create a new configuration tree that is associated with the ID of `2957FCA1FE3B3C`. This ID is used for any further interaction with this configuration session.

 **Note:** The API does not place any limits on the number of configuration sessions or background processes that can be created. Clients are responsible for process management and should release resources that are not in use.

The example configuration mode session that is shown here includes the following events.

- A request is made to view information about active configuration sessions to see that the new session is opened.
- Configuration information is then modified, committed, and saved.
- The session is ended.
- Active configuration is viewed again to ensure that the session was removed.

**Table 4. A configuration mode session**

Description	Command and Response
<p>Display the active configuration sessions. The only difference between this example and the previous example is that the POST command is replaced with the GET command.</p>	<pre>GET / rest/conf Host:  10.0.0.23 2 Accept:  applicati  on/json Vyatta-Spe  cification -Version:  0.1 Authorizat  ion: Basic  dnlhdHRhO  nZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded. In this case, the output (in JSON format) shows that two active configuration sessions are associated with the user that initiated the command (user "vyatta") and each session has a unique ID. Note that the "started" and "updated" fields display the time the session started and the time it was last updated in UNIX epoch time (the number of seconds since January 1, 1970). The "modified" field indicates whether uncommitted changes exist in the configuration session.</p>	<pre>HTTP/1.1 200 OK Content-Ty  pe:  applicati  on/json Content-Le  ngth: 412 Date:  Thu, 06  May 2010  03:15:18  GMT Server:  lighttpd/  1.4.19  {  "session"  : [  {  "id":  "2957FC9C  AlFE3B3C",  "username"  :  "vyatta",  "descript  ion": "",  "started"  :  "12730855  68",  "modified"  :  "false",  "updated"  :  "12730855  68"  },  {  "id":  "F8430722  2F469A8A",  "username"  :  "vyatta",  "descript  ion": "",  "started"  :</pre>



Description	Command and Response
	<pre>"12730853 37", "modified ": "false", "updated" : "12730853 37" } ], "message" : " " }</pre>
<p>Change the SSH port to 1011. The request is equivalent to the <code>set service ssh port 1011</code> command in the CLI configuration mode. Note that the URI uses the configuration session ID that was created previously.</p>	<pre>PUT / rest/conf/ 2957FC9CA1 FE3B3C/set /service/s sh/port/10 11 Host: 10.0.0.23 2 Accept: applicati on/json Vyatta-Spe cification -Version: 0.1 Authorizat ion: Basic dnlhdHRhO nZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded.</p>	<pre>HTTP/1.1 200 OK Transfer-E ncoding: chunked Content-Ty pe: applicati on/json Date: Thu, 06 May 2010 03:48:46 GMT Server: lighttpd/ 1.4.19</pre>
<p>Commit the configuration change. This request is equivalent to the <code>commit</code> command in the CLI configuration mode.</p>	<pre>POST / rest/conf/ 2957FC9CA1 FE3B3C/com mit Host: 10.0.0.23 2 Accept: applicati on/json Vyatta-Spe cification -Version: 0.1 Authorizat ion: Basic dnlhdHRhO nZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded. Note that the SSH daemon (sshd) was restarted because of the configuration change.</p>	<pre>HTTP/1.1 200 OK</pre>

Description	Command and Response
	<pre>Content-Type:   application/json Content-Length: 69 Date:   Thu, 06   May 2010   03:51:13   GMT Server:   lighttpd/   1.4.19  {   "message"   : "   Restarting OpenBSD   Secure   Shell   server:   sshd.\n " }</pre>
<p>Save the active configuration. This request is equivalent to the <code>save</code> command in CLI configuration mode.</p>	<pre>POST / rest/conf/ 2957FC9CA1 FE3B3C/save Host:   10.0.0.23   2 Accept:   application/json Vyatta-Specification -Version:   0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded. The message shows where the file was saved.</p>	<pre>HTTP/1.1 200 OK Content-Type:   application/json Content-Length: 94 Date:   Thu, 06   May 2010   03:55:24   GMT Server:   lighttpd/   1.4.19  {   "message"   : " Saving   configura   tion to   '/config   /config.bo   ot'...\n   Done\n " }</pre>
<p>Terminate the configuration session. This request is equivalent to the <code>exit</code> command in CLI configuration mode.</p>	<pre>DELETE / rest/conf/ 2957FC9CA1 FE3B3C Host:   10.0.0.23   2</pre>

Description	Command and Response
	<pre>Accept: applicati on/json Vyatta-Spe cification -Version: 0.1 Authorizat ion: Basic dnlhdHRhO nZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded.</p>	<pre>HTTP/1.1 200 OK Content-Ty pe: applicati on/json Content-Le ngth: 21 Date: Thu, 06 May 2010 03:59:12 GMT Server: lighttpd/ 1.4.19  {  "message" : " " } }</pre>
<p>Display the active configuration sessions.</p>	<pre>GET / rest/conf Host: 10.0.0.23 2 Accept: applicati on/json Vyatta-Spe cification -Version: 0.1 Authorizat ion: Basic dnlhdHRhO nZ5YXR0YQ= =</pre>
<p>System response. The 200 response code indicates that the request succeeded. In this case, the session that is terminated no longer appears in the list of active configuration sessions. A single active configuration session remains.</p>	<pre>HTTP/1.1 200 OK Content-Ty pe: applicati on/json Content-Le ngth: 226 Date: Thu, 06 May 2010 03:59:30 GMT Server: lighttpd/ 1.4.19  {  "session" : [ {  "id": "F8430722 2F469A8A", "username" : "vyatta",</pre>

Description	Command and Response
	<pre>"description": "",  "started" : "1273085337",  "modified" : "false",  "updated" : "1273085337" } ],  "message" : " }</pre>

# Chapter 6. Configuration Mode Command Reference

---

## **DELETE /rest/conf/<conf-id>**

Terminates a configuration mode session.

### **Syntax**

CLI equivalent: `exit`

`DELETE /rest/conf/conf-id`

### **Mode**

Configuration mode

### **Parameters**

*conf-id*

The ID of a configuration session to terminate.

### **Request and Response Content**

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Command Response

Response Status: 200, 400, 401, 403, 404

### **Usage Guidelines**

Use this command to terminate a configuration mode session, removing the session ID and releasing system resources that are associated with the session.

### **Examples**

#### **Request**

The following example ends a configuration session on 10.0.0.232.

```
DELETE /rest/conf/27755B4BC823272E
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
```

```
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 21
Date: Fri, 19 Feb 2010 21:55:21 GMT
Server: lighttpd/1.4.19
```

```
{
  "message": " "
  "error": " "
}
```

---

## GET /rest/conf

Displays a list of the active configuration sessions for the host at an IP address.

### Syntax

CLI equivalent: None.

GET **/rest/conf**

### Mode

Configuration mode.

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Session data

Response Status: 200, 400, 401, 403, 404

### Usage Guidelines

Use this command to display a list of the active configuration sessions for the host at an IP address.

The JSON-formatted response displays only sessions that are associated with the user and the start and last-modified times (in UNIX epoch time). The “modified” key is a Boolean value that shows whether local (that is, uncommitted) changes exist in the configuration session.

## Examples

### Request

The following example retrieves a list of active configuration sessions. The `GET` command requests a list of all active configuration sessions on 10.0.0.232. The response contains the session ID in the Location field.

```
GET /rest/conf
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

The following example requests a list of all active configuration sessions on 10.0.0.232.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 523
Date: Fri, 19 Feb 2010 21:50:51 GMT
Server: lighttpd/1.4.19

{
  "message": " ",
  "session": [
    {
      "id": "27755B4BC823272E",
      "username": "vyatta",
      "started": "1266616149",
      "modified": "true",
      "updated": "1266616149",
      "description": ""
    },
    {
      "id": "F13BF3B55FE72DF3",
      "username": "vyatta",
      "started": "1266615998",
      "modified": "false",
      "updated": "1266615998",
      "description": "firewall-work"
    },
    {
      "id": "DE54D909FA4047B2",
      "username": "vyatta",
      "started": "1266614404",
      "modified": "false",
      "updated": "1266614404",
      "description": "current-configuration"
    }
  ]
}
```

## GET /rest/conf/<conf-id>/<path>

Returns the definition of a configuration mode parameter, its current value, and a list of its children.

### Syntax

CLI equivalent: `show` (in configuration mode)

GET `/rest/conf/conf-id/path`

### Mode

Configuration mode

### Parameters

#### *conf-id*

The configuration session ID that is associated with the configuration session.

#### *path*

The path to the configuration node.

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Configuration data

Response Status: 200, 400, 401, 403, 404

### Usage Guidelines

Use this command to display the definition of a configuration mode parameter, its current value, and a list of its children.

As in the CLI, the user must create a configuration session before parameters can be modified. The response is formatted as JSON and returned as a hash. The current node may return name, state, type, enumeration, end, mandatory, multi, default, help, val\_help, comp\_help, and children.

A brief description of each of these fields follows:

**Name:** Actual name of this node. The name is the same as the last switch of the request URI.

**State:** Can be [active, set, delete, none]. Whether this node is active or not active in the current configuration, or has been modified in the local configuration only (pending commit or discard).

**Type:** For value nodes. Can be [none, bool, text, ipv4, ipv6, ipv4net, ipv6net, macaddr, u32].



Enumeration: List of allowed values for this node.

End: Whether this node is the last element of the tree.

Mandatory: Whether this node is required in the configuration.

Multi: Whether this node can be set to more than one value.

Default: The default value, if this node has one, that is returned.

Help: Help text.

Val\_help: Whether the node supports more than one type of input or if the completion help string needs to be different than the primary help string.

Comp\_help: "Detailed Information" help text.

Children: Array of configurable children under this node with their current state in the local and active configuration tree.

## Examples

### Request

The following example retrieves configuration information for the service ssh node. The request specifies the configuration parameter definitions for the service ssh configuration node on 10.0.0.232 with session ID 27755B4BC823272E.

```
GET /rest/conf/27755B4BC823272E/service/ssh
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 413
Date: Fri, 19 Feb 2010 21:47:03 GMT
Server: lighttpd/1.4.19
```

```
{
  "children": [
    {
      "name": "protocol-version",
      "state": "active"
    },
    {
      "name": "port",
      "state": "active"
    },
    {
      "name": "allow-root",
      "state": "none"
    }
  ]
}
```

```

    "name": "disable-password-authentication",
    "state": "none"
  }
],
"help": " Enable/disable Secure SHell (SSH)
protocol",
"name": "ssh",
"state": "active",
"type": "NONE"
}

```

## POST /rest/conf

Creates a new configuration session ID that provides access to configuration mode commands.

### Syntax

CLI equivalent: `configure` (in operational mode).

POST **/rest/conf**[/*description*]

### Mode

Configuration mode.

### Parameters

#### *description*

A descriptive tag for the session.

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type, Cookie

Response Message Body: N/A

Response Status: 200, 201, 400, 401, 403, 404

### Usage Guidelines

Use this command to create a new configuration session ID that provides access to configuration mode commands; this command is equivalent to using a `show` command in the CLI configuration mode.

The client must create a configuration session before any operations can be performed on configuration information; this session is equivalent to entering configuration mode in the CLI.

When the session is created, a configuration session ID is created and returned within the response header in the Location parameter. (In the following example, the session ID is 27755B4BC823272E.) The configuration session ID is globally unique.

Note that configuration sessions that are created by using the API persist indefinitely (even across reboots) unless explicitly deleted. You should make sure that you delete configuration resources when the session is finished to free associated resources.

A created configuration session can be associated with an optional description field. This description can be used in place of the configuration session ID in referencing a configuration session with the other configuration commands.

## Examples

### Request

The following example creates a new configuration session on 10.0.0.232. In the response, the session ID is returned in the Location field.

```
POST /rest/conf
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 201 OK
Transfer-Encoding: chunked
Content-Type: application/json
Location: rest/conf/27755B4BC823272E
Date: Fri, 19 Feb 2010 21:49:09 GMT
Server: lighttpd/1.4.19
```

### Request

The following example creates a new configuration session by using a description. The request creates a configuration session on 10.0.0.232 using the string “XYZ” as a description string. In the response, the session ID is returned in the Location field.

```
POST /rest/conf/XYZ
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 201 OK
Transfer-Encoding: chunked
Content-Type: application/json
Location: rest/conf/38755B4BC823273F
Date: Fri, 19 Feb 2010 21:52:23 GMT
Server: lighttpd/1.4.19
```

## Request

The following example requests a list of all active configuration sessions on 10.0.0.232. In the response, the session ID is displayed in the id field and the description string is displayed in the description field. The description string can be used instead of the session ID in all commands that use the session ID.

```
GET /rest/conf
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 523
Date: Fri, 19 Feb 2010 21:52:38 GMT
Server: lighttpd/1.4.19
```

```
{
  "message": " ",
  "session": [
    {
      "id": "38755B4BC823273F",
      "username": "vyatta",
      "started": "1266616163",
      "modified": "false",
      "updated": "1266616163",
      "description": "XYZ"
    }
  ]
}
```

## Request

The following example ends a configuration session on 10.0.0.232 using the session description rather than the session ID.

```
DELETE /rest/conf/XYZ
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 21
Date: Fri, 19 Feb 2010 21:54:12 GMT
Server: lighttpd/1.4.19
```

```
{
  "message": " "
```

```
"error": " "
}
```

## POST /rest/conf/<conf-id>/<cmd>

Runs an action command in configuration mode.

### Syntax

CLI equivalent: `commit`, `save`, `load`, `merge`, `show`.

POST `/rest/conf/conf-id/cmd[/path]`

### Mode

Configuration mode.

### Parameters

#### *conf-id*

The configuration session ID that is associated with the configuration session.

#### *cmd*

The configuration action command to run.

#### *path*

The path to the configuration node to be affected by the command. The path is equivalent to the parameters for the CLI configuration mode.

### Request and Response Content

Request Headers: `Accept`, `Authorization`, `Vyatta-Specification-Version`

Request Message Body: N/A


Response Headers: `Content-Length`, `Content-Type`, `Cookie`

Response Message Body: N/A

Response Status: 200, 400, 401, 403, 404

### Usage Guidelines

Use this command to run an action command in configuration mode. Action commands are configuration commands that do not modify local configuration nodes in the way, for example, that the `set`, `delete`, and `run` commands do. The `commit`, `save`, `load`, `discard`, `merge`, and `show` commands are supported.

 **Note:** To specify `show -all`, use `show-all` (no space) in the path.

The `edit`, `exit`, `copy`, and `rename` commands are not supported.

You must create a configuration session before parameters can be modified; this session is equivalent to entering configuration mode in the CLI. In addition, as in the CLI, changes must be committed for them to be applied to the system.

## Examples

### Request

The following example commits changes on 10.0.0.232. In the response, no changes were made so none were committed.

```
POST /rest/conf/27755B4BC823272E/commit
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 57
Date: Fri, 19 Feb 2010 21:50:00 GMT
Server: lighttpd/1.4.19
```

```
{
  "message": "No configuration changes to
commit\n "
}
```

---

## PUT rest/conf/<conf-id>/delete/<path>

Deletes configuration information.

### Syntax

CLI equivalent: `delete`.

PUT **rest/conf/conf-id/delete/path**

### Mode

Configuration mode

### Parameters

#### *conf-id*

The configuration session ID that is associated with the configuration session.

#### *path*

The path to the configuration node. The path is equivalent to the parameters for the CLI configuration mode.

## Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type


Response Message Body: Command Response

Response Status: 200, 400, 401, 403, 404

## Usage Guidelines

Use this command to delete configuration information. This command is equivalent to the `delete` command in the CLI.

Note that special characters (that is, characters that are not valid HTTP URL characters—for example, spaces) must be encoded within the URL character string. The API automatically converts the encoded characters back to the intended characters.

 **Note:** Characters that are not valid URL characters have to be URL encoded (for example, spaces). These encodings are reconverted on the router.

Responses to this command may return messages that are associated with this command in the response body. All commands are local to the session until the API command that represents the `commit` action is applied.

## Examples

### Request

The following example deletes the SSH service on 10.0.0.232.

```
PUT /rest/conf/27755B4BC823272E/delete/service/ssh
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json
Date: Fri, 19 Feb 2010 21:52:12 GMT
Server: lighttpd/1.4.19
```

---

## PUT /rest/conf/<conf-id>/set/<path>

Sets configuration information.

## Syntax

CLI equivalent: `set`

`PUT /rest/conf/conf-id/set/path`

## Mode

Configuration mode

## Parameters

### *conf-id*

The configuration session ID that is associated with the configuration session.

### *path*

The path to the configuration node. The path is equivalent to the parameters for the CLI configuration mode.

## Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Command Response

Response Status: 200, 400, 401, 403, 404

## Usage Guidelines

Use this command to set configuration information. This command is equivalent to the `set` command in the CLI.

Note that special characters (that is, characters that are not valid HTTP URL characters—for example, spaces) must be encoded within the URL character string. The API automatically converts the encoded characters back to the intended characters.

This command may return messages that are associated with this command in the response body. All commands are local to the session until the API command that represents the commit action is applied.

## Examples

### Request

The following example changes the SSH port on 10.0.0.232.

```
PUT /rest/conf/27755B4BC823272E/set/service/ssh/port/22
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
```



```
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK  
Transfer-Encoding: chunked  
Content-Type: application/json  
Date: Fri, 19 Feb 2010 21:52:12 GMT  
Server: lighttpd/1.4.19
```

---

## Chapter 7. Operational Mode

This chapter describes the functionality that is available in the operational mode of the router Remote Access API 2.0.


---

### Overview


Operational mode provides the ability to run operational mode commands remotely and display their output. Each command initiates a process that may have a finite life span (for example, `show version`) or may run indefinitely until explicitly stopped (for example, `ping address`).

Commands with finite life spans continue to consume system resources until the client either requests that the resources be released (using `DELETE`) or it finishes reading the output that is generated by the command. Receiving a response code of 410 (GONE) indicates that the output has been consumed. Output that is not read remains on the system until the next reboot or until the HTTPS service is restarted.

Commands with infinite life spans must be stopped explicitly (using `DELETE`). Output is removed when the process is deleted or the output has been completely read.

 **Note:** Operational mode commands that are interactive in nature (for example, `add system image`) are not supported in the API.

Operational mode requests use **rest/op** or **rest/op/op-id** as part of the URI (as opposed to **rest/conf** or **rest/conf/session-id** used in configuration mode).

 **Note:** Restarting the HTTPS service invalidates operational session IDs, but does not invalidate configuration session IDs.

---

### Summary of tasks

You can perform the following tasks by using the commands that are described in this chapter.

Execute an operational mode command.	<a href="#">POST /rest/op/&lt;path&gt;</a>
Retrieve operational mode parameter definitions.	<a href="#">GET /rest/op/&lt;path&gt;</a>
Display a list of operational mode processes.	<a href="#">GET /rest/op</a>
Retrieve output.	<a href="#">GET /rest/op/&lt;process-id&gt;</a>
Terminate an operational mode process.	<a href="#">DELETE /rest/op/&lt;process-id&gt;</a>

## Example of one-time output

The following table shows a request to a router at 10.0.0.232 to display the system version information. The request is equivalent to the `show version` operational mode command. This command has a finite life span and the system resources that it uses are freed after its output is read.

**Table 5. Generating a request to display version information**

Description	Command and Response
Display system version information. Note that <code>rest/op</code> is used rather than <code>rest/conf</code> . Note also that a session is not begun as is done in configuration mode.	<pre>POST /rest/op/show/version Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==</pre>
System response. The 201 response code indicates that the request succeeded and that the command output is available at the specified location. To display the output, a separate request must be submitted that specifies the output location.	<pre>HTTP/1.1 201 Created Transfer-Encoding: chunked Content-Type: application/json Location: rest/op/137AA3B22A362CA3 Date: Thu, 06 May 2010 04:26:08 GMT Server: lighttpd/1.4.19</pre>
Display the output from the <code>show version</code> command. Note that the GET command is used and that the previous location information is used in the URI.	<pre>GET /rest/op/137AA3B22A362CA3 Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==</pre>
System response. The 200 response code indicates that the request succeeded. The output from the <code>show version</code> command is returned in plain text format.	<pre>HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 329 Date: Thu, 06 May 2010 04:26:44 GMT Server: lighttpd/1.4.19  Version: 999.larkspur.04280031 Description: 999.larkspur.04280031 Copyright: 2006-2010 Vyatta, Inc. Built by: autobuild@vyatta.com Built on: Wed Apr 28 07:31:19 UTC 2010 Build ID: 1004280731-ff5e5c7 Boot via: image Uptime: 04:26:09 up 7 days, 1:09, 2 users, load average: 0.00, 0.00, 0.00</pre>
Display the output from the <code>show version</code> command a second time. This time the result is different.	<pre>GET /rest/op/137AA3B22A362CA3 Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==</pre>
System response. The 410 response code indicates that the request failed and that the output is "gone" as it was "consumed" by the first request. All system resources used by the initial command have been freed.	<pre>HTTP/1.1 410 Gone Content-Type: application/json Content-Length: 0 Date: Thu, 06 May 2010 04:26:54 GMT Server: lighttpd/1.4.19</pre>

## Example of continuous output

The previous example shows a command that generates its output and finishes (finite life span). The following example shows a command that continues to generate output until it is explicitly stopped (infinite life span). The client may need to make several requests for data before the 410 response is received (if the process generating the output terminates). Each response is a sequential piece of output from the command that is run. If the process that is generating the output does not terminate, output can be retrieved only while the process is running. After the process is stopped, the output is deleted.

The example shows a request to a router at 10.0.0.232 to ping another device. The request is equivalent to the `ping address` operational mode command.

**Table 6. Generating a request to ping a device**

Description	Command and Response
Ping the device at 10.0.0.1 from 10.0.0.232.	<pre>POST / rest/op/ping/10 .0.0.1 Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ==</pre>
System response. The 201 response code indicates that the request succeeded and that the command output is available at the specified location. To display the output, a separate request must be submitted that specifies the output location.	<pre>HTTP/1.1 201 Created Transfer-Encodi ng: chunked Content-Type: application/js on Location: rest/op/02B347 9CA1522F2A Date: Fri, 07 May 2010 17:10:30 GMT Server: lighttpd/1.4.1 9</pre>
Display the output from the <code>ping</code> command. Note that the <code>GET</code> command is used and that the previous location information is used in the URI.	<pre>GET / rest/op/02B3479 CA1522F2A Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ==</pre>
System response. The 200 response code indicates that the request succeeded. The output from the <code>ping</code> command is returned in plain text format. The <code>ping</code> command continues to run in the background and generate output.	<pre>HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 1164 Date: Fri, 07 May 2010 17:10:49 GMT</pre>

**Table 6. Generating a request to ping a device (continued)**

Description	Command and Response
	<pre> Server: lighttpd/1.4.1 9  PING 10.3.0.1 (10.3.0.1) 56(84) bytes of data. 64 bytes from 10.3.0.1: icmp_seq=1 ttl=64 time=0.839 ms 64 bytes from 10.3.0.1: icmp_seq=2 ttl=64 time=0.846 ms 64 bytes from 10.3.0.1: icmp_seq=3 ttl=64 time=0.787 ms 64 bytes from 10.3.0.1: icmp_seq=4 ttl=64 time=0.844 ms 64 bytes from 10.3.0.1: icmp_seq=5 ttl=64 time=0.850 ms 64 bytes from 10.3.0.1: icmp_seq=6 ttl=64 time=0.791 ms 64 bytes from 10.3.0.1: icmp_seq=7 ttl=64 time=0.836 ms 64 bytes from 10.3.0.1: icmp_seq=8 ttl=64 time=0.910 ms 64 bytes from 10.3.0.1: icmp_seq=9 ttl=64 time=0.861 ms 64 bytes from 10.3.0.1: icmp_seq=10 ttl=64 time=0.823 ms 64 bytes from 10.3.0.1: icmp_seq=11 ttl=64 time=0.857 ms 64 bytes from 10.3.0.1: icmp_seq=12 ttl=64 time=0.823 ms 64 bytes from 10.3.0.1: icmp_seq=13 ttl=64 time=0.791 ms 64 bytes from 10.3.0.1: icmp_seq=14 ttl=64 time=0.806 ms 64 bytes from 10.3.0.1: icmp_seq=15 ttl=64 time=0.831 ms 64 bytes from 10.3.0.1:                     </pre>

**Table 6. Generating a request to ping a device (continued)**

Description	Command and Response
	<pre> icmp_seq=16 ttl=64 time=0.811 ms 64 bytes from 10.3.0.1: icmp_seq=17 ttl=64 time=0.800 ms 64 bytes from 10.3.0.1: icmp_seq=18 ttl=64 time=0.821 ms </pre>
Display the output from the ping command a second time. This time the result is different.	<pre> GET / rest/op/02B3479 CA1522F2A Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ== </pre>
System response. The 200 response code indicates that the request succeeded. Notice that the output is a continuation from the previous request. The ping command continues to run in the background until it is terminated.	<pre> HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 248 Date: Fri, 07 May 2010 17:10:52 GMT Server: lighttpd/1.4.1 9  64 bytes from 10.3.0.1: icmp_seq=19 ttl=64 time=0.799 ms 64 bytes from 10.3.0.1: icmp_seq=20 ttl=64 time=0.807 ms 64 bytes from 10.3.0.1: icmp_seq=21 ttl=64 time=0.753 ms 64 bytes from 10.3.0.1: icmp_seq=22 ttl=64 time=0.798 ms </pre>
Display the list of active operational mode processes.	<pre> GET /rest/op Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ== </pre>
System response. The 200 response code indicates that the request succeeded. The output (in JSON format) displays the requested information that shows a single active operational mode process.	<pre> HTTP/1.1 200 OK Content-Type: application/js on </pre>

**Table 6. Generating a request to ping a device (continued)**

Description	Command and Response
	<pre>Content-Length: 193 Date: Fri, 07 May 2010 17:10:58 GMT Server: lighttpd/1.4.1 9 {   "process": [     {       "username":       "vyatta",       "started":       "1273252231",       "updated":       "8",       "id":       "02B3479CA1522       F2A",       "command":       "ping       10.0.0.1"     }   ] }</pre>
<p>Stop an active operational mode process. Note that the DELETE command is used and the previous location information, which contains the unique process ID in the operational mode, is used in the URI. This request stops the ping command and removes any remaining output, freeing up all resources used by the command.</p>	<pre>DELETE / rest/op/02B3479 CA1522F2A Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ==</pre>
<p>System response. The 200 response code indicates that the request succeeded.</p>	<pre>HTTP/1.1 200 OK Content-Type: application/js on Content-Length: 21 Date: Fri, 07 May 2010 17:11:20 GMT Server: lighttpd/1.4.1 9 {   "message": " " }</pre>
<p>Display the list of active operational mode processes.</p>	<pre>GET /rest/op Host: 10.0.0.232 Accept: application/js on Vyatta-Specific ation-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YX R0YQ==</pre>

**Table 6. Generating a request to ping a device (continued)**

Description	Command and Response
System response. The 200 response code indicates that the request succeeded, but no active operational processes are present as the response body is empty.	HTTP/1.1 200 OK Content-Type: application/javascript Content-Length: 0 Date: Fri, 07 May 2010 17:39:46 GMT Server: lighttpd/1.4.19



# Chapter 8. Operational Mode Command Reference

---

## **DELETE /rest/op/<process-id>**

Terminates an operational mode process.

### **Syntax**

`DELETE /rest/op/process-id`

### **Mode**

Operational Mode

### **Parameters**

*process-id*

The ID of an operational mode process to terminate.

### **Request and Response Content**

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Command Response

Response Status: 200, 400, 401, 403, 404

### **Usage Guidelines**

Use this command to terminate an operational mode process and release the system resources that are associated with it.

### **Examples**

#### **Request**

The following example requests to terminate the operational mode process identified by the specified process ID on 10.0.0.232.

```
DELETE /rest/op/D90078870BEB3FF5
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 21
Date: Fri, 19 Feb 2010 21:45:30 GMT
Server: lighttpd/1.4.19

{
  "message": " "
  "error": " "
}

```

## GET /rest/op

Displays a list of the operational mode commands that are still running, have pending data output, or both.

### Syntax

GET /rest/op

### Mode

Operational Mode

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Op mode process data.

Response Status: 200, 202, 400, 401, 403, 404

### Usage Guidelines

Use this command to display a list of the operational mode commands that are still running, have pending data output, or both.

Processes that are displayed in this list are taking up system resources and should be removed from the system if they are no longer needed.

Data returned in the response body is an array of process hashes. Only processes that are initiated by the user as specified in the authentication header are displayed. Time values (start and update) are displayed in UNIX epoch time (the number of seconds since January 1, 1970).

## Examples

### Request

The following example uses the GET option to retrieve a list of active operational mode commands on 10.0.0.232.

```
GET /rest/op
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 1080
Date: Fri, 19 Feb 2010 21:41:27 GMT
Server: lighttpd/1.4.19
```

```
{
  "process": [
    {
      "username": "vyatta",
      "started": "1266614867",
      "updated": "1919251319",
      "id": "23BA16677B8D8D4C",
      "command": "show users"
    },
    {
      "username": "vyatta",
      "started": "1266614435",
      "updated": "1919251319",
      "id": "83FDE523A1548B5E",
      "command": "show tech-support"
    },
    {
      "username": "vyatta",
      "started": "1266615495",
      "updated": "1919251319",
      "id": "A12A9F9707621658",
      "command": "show interfaces dataplane
dp0plp1"
    },
    {
      "username": "vyatta",
      "started": "1266614874",
      "updated": "1919251319",
      "id": "D90078870BEB3FF5",
      "command": "show users"
    }
  ]
}
```

## GET /rest/op/<path>

Displays the definition of an operational mode command and a list of its children.

### Syntax

```
GET /rest/op/path
```

### Mode

Operational Mode

### Parameters

*path*

The path to an operational command.

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Op mode configuration data.

Response Status: 200, 202, 400, 401, 403, 404

### Usage Guidelines

Use this command to display the definition of an operational mode command and a list of its children.

Data is returned in a JSON hash. The data includes help for the node that is being requested, a Boolean value indicating whether this command can be run, and a list of the node children and enumerated values, if available.

### Examples

#### Request

The following example shows how to use the GET option to retrieve parameter definitions in operational mode for the dp0p1p1 data plane interface on 10.0.0.232.

```
GET /rest/op/show/interfaces/dataplane/dp0p1p1
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
```

```

Content-Type: application/json
Content-Length: 257
Date: Fri, 19 Feb 2010 21:32:32 GMT
Server: lighttpd/1.4.19

{
  "children": [
    "brief",
    "capture",
    "identify",
    "physical",
    "queue",
    "statistics",
    "vif"
  ],
  "enum": [
    "dp0plp1",
    "dp0plp2",
    "dp0plp3"
  ],
  "action": "true",
  "help": " Show specified dataplane interface
information"
}

```

---

## GET/rest/op/<process-id>

Displays the output from an operational mode command.

### Syntax

GET/rest/op/*process-id*

### Mode

Operational mode

### Parameters

***process-id***

The process ID that is used to display operational mode command output.

### Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: Command Response

Response Status: 200, 202, 400, 401, 403, 404, 410

## Usage Guidelines

Use this command to display the output from an operational command.

Note that some commands may not terminate by themselves. The developer is responsible for managing nonterminating commands (processes). Note also that sometimes queries can be generated faster than the command can produce data; in this case, the command displays a 200 or 202 status code. The client at this point can continue to request data from this process until a 410 response is received. Commands that display a 410 response do not require any further process management or client-initiated process deletion.

The response is plain text and not JSON.

## Examples

### Request

```
GET /rest/op/A12A9F9707621658
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 579
Date: Fri, 19 Feb 2010 21:39:32 GMT
Server: lighttpd/1.4.19

dp0p1p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:d3:1b:7a brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.232/24 brd 10.3.0.255 scope global dp0p1p1
    inet6 fe80::20c:29ff:fed3:1b7a/64 scope link
        valid_lft forever preferred_lft forever

    RX: bytes  packets  errors  dropped  overrun  mcast
          233008    1179      0        0         0         0
    TX: bytes  packets  errors  dropped  carrier
collisions
          187036    543        0         0         0         0
```

## POST /rest/conf

Creates a new configuration session ID that provides access to configuration mode commands. (CLI equivalent of `configure` command in operational mode)

## Syntax

POST **/rest/conf**[*/description*]

## Mode

Configuration mode

## Parameters

### *description*

Optional. A descriptive tag for the session.

## Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type, Cookie

Response Message Body: N/A

Response Status: 200, 201, 400, 401, 403, 404

## Usage Guidelines

Use this command to obtain configuration information; this is the equivalent of using a show command in the CLI configuration mode.

The client must create a configuration session before any operations can be performed on configuration information; this is the equivalent of entering configuration mode in the CLI.

When the session is created, a configuration session ID is created and returned within the response header in the Location: parameter. (In the example, the session ID is "27755B4BC823272E.") The configuration session ID is globally unique.

Note that configuration sessions created using the API persist indefinitely (even across reboots) unless explicitly deleted. You should make sure you delete configuration resources when the session is finished to free associated resources.

A created configuration session can be associated with an optional description field. This description can be used in place of the configuration session ID in referencing a configuration session with the other configuration commands.

## Examples

### Request

The following example creates a new configuration session on 10.0.0.232. Note the session ID in the Location field in the system response.

```
POST /rest/conf
Host: 10.0.0.232
```

```
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 201 OK
Transfer-Encoding: chunked
Content-Type: application/json
Location: rest/conf/27755B4BC823272E
Date: Fri, 19 Feb 2010 21:49:09 GMT
Server: lighttpd/1.4.19
```

## Request

The following example creates a new configuration session on 10.0.0.23 using the string “XYZ” as a description string. Again the session ID is returned in the Location field in the system response.

```
POST /rest/conf/XYZ
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 201 OK
Transfer-Encoding: chunked
Content-Type: application/json
Location: rest/conf/38755B4BC823273F
Date: Fri, 19 Feb 2010 21:52:23 GMT
Server: lighttpd/1.4.19
```

---

## POST/rest/op/<path>

Runs an operational mode CLI command on the remote system.

### Syntax

```
POST/rest/op/path
```

### Mode

Operational mode.

### Parameters

#### *path*

The path to an operational mode command.



## Request and Response Content

Request Headers: Accept, Authorization, Vyatta-Specification-Version

Request Message Body: N/A

Response Headers: Content-Length, Content-Type

Response Message Body: N/A

Response Status: 201, 400, 401, 403, 404

## Usage Guidelines

Use this command to run an operational mode CLI command on the remote system.

Running this command initiates an asynchronous process that performs the command.

Note that a success response (2xx) does not guarantee successful execution of the command. The response header on success (HTTP status code 201) contains the path (in the Location: parameter) to the command response, which must be used in any subsequent GET command to obtain any data associated with this command, and command success or failure.

The Location: parameter identifies the resource location. Another GET command that is entered for this location displays any results that are associated with the command. However, as the command may still be running, successive GET commands may be required to obtain the full response (refer to [Example of continuous output](#) for details).

## Examples

### Request

```
POST
/rest/op/clear/interfaces/dataplane/dp0plp1/counters
Host: 10.0.0.232
Accept: application/json
Vyatta-Specification-Version: 0.1
Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
```

```
HTTP/1.1 201 Created
Transfer-Encoding: chunked
Content-Type: application/json
Location: rest/op/3479DEF17C6AF4D1
Date: Fri, 19 Feb 2010 21:27:57 GMT
Server: lighttpd/1.4.19
```

## Chapter 9. HTTP Status Codes

This appendix lists the HTTP status codes that are returned by the router.

**Table 7. HTTP status codes**

Command	Status Code	Description
General		
	400 Bad Request	Client request error (body exceeded limit)
	401 Unauthorized	Authorization error
Operational Mode		
	400 Bad Request	Client request error (malformed request)
• DELETE		
	400 Bad Request	Client request error
	500 Internal Server Error	Server side error
• GET		
	200 OK	Returned data
	202 Accepted	Running process, but no new data
	404 Not Found	Undetected configuration data
	410 Gone	End of data retrieval for the process
	500 Internal Server Error	Server request error
• POST		
	201 Created	Process initiation
	400 Bad Request	Client request error
	500 Internal Server Error	Server side error
Configuration Mode		
	400 Bad Request	Client request error (malformed request)
• DELETE		
	400 Bad Request	Client request error
• GET		
	404 Not Found	Undetected configuration data
	500 Internal Server Error	Server side error
• POST		
	201 Created	Creation of configuration resource
	400 Bad Request	Client request error
	500 Internal Server Error	Server side error
• PUT		
	400 Bad Request	Client request error

## Chapter 10. List of Acronyms

Acronym	Description
ACL	access control list
ADSL	Asymmetric Digital Subscriber Line
AH	Authentication Header
AMI	Amazon Machine Image
API	Application Programming Interface
AS	autonomous system
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BGP	Border Gateway Protocol
BIOS	Basic Input Output System
BPDU	Bridge Protocol Data Unit
CA	certificate authority
CCMP	AES in counter mode with CBC-MAC
CHAP	Challenge Handshake Authentication Protocol
CLI	command-line interface
DDNS	dynamic DNS
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol version 6
DLCI	data-link connection identifier
DMI	desktop management interface
DMVPN	dynamic multipoint VPN
DMZ	demilitarized zone
DN	distinguished name
DNS	Domain Name System
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
eBGP	external BGP
EBS	Amazon Elastic Block Storage
EC2	Amazon Elastic Compute Cloud
EGP	Exterior Gateway Protocol
ECMP	equal-cost multipath
ESP	Encapsulating Security Payload
FIB	Forwarding Information Base
FTP	File Transfer Protocol

Acronym	Description
GRE	Generic Routing Encapsulation
HDLCL	High-Level Data Link Control
I/O	Input/Output
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IGP	Interior Gateway Protocol
IPS	Intrusion Protection System
IKE	Internet Key Exchange
IP	Internet Protocol
IPOA	IP over ATM
IPsec	IP Security
IPv4	IP Version 4
IPv6	IP Version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISM	Internet Standard Multicast
ISP	Internet Service Provider
KVM	Kernel-Based Virtual Machine
L2TP	Layer 2 Tunneling Protocol
LACP	Link Aggregation Control Protocol
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LLDP	Link Layer Discovery Protocol
MAC	medium access control
mGRE	multipoint GRE
MIB	Management Information Base
MLD	Multicast Listener Discovery
MLPPP	multilink PPP
MRRU	maximum received reconstructed unit
MTU	maximum transmission unit
NAT	Network Address Translation
NBMA	Non-Broadcast Multi-Access
ND	Neighbor Discovery
NHRP	Next Hop Resolution Protocol
NIC	network interface card

Acronym	Description
NTP	Network Time Protocol
OSPF	Open Shortest Path First
OSPFv2	OSPF Version 2
OSPFv3	OSPF Version 3
PAM	Pluggable Authentication Module
PAP	Password Authentication Protocol
PAT	Port Address Translation
PCI	peripheral component interconnect
PIM	Protocol Independent Multicast
PIM-DM	PIM Dense Mode
PIM-SM	PIM Sparse Mode
PKI	Public Key Infrastructure
PPP	Point-to-Point Protocol
PPPoA	PPP over ATM
PPPoE	PPP over Ethernet
PPTP	Point-to-Point Tunneling Protocol
PTMU	Path Maximum Transfer Unit
PVC	permanent virtual circuit
QoS	quality of service
RADIUS	Remote Authentication Dial-In User Service
RHEL	Red Hat Enterprise Linux
RIB	Routing Information Base
RIP	Routing Information Protocol
RIPng	RIP next generation
RP	Rendezvous Point
RPF	Reverse Path Forwarding
RSA	Rivest, Shamir, and Adleman
Rx	receive
S3	Amazon Simple Storage Service
SLAAC	Stateless Address Auto-Configuration
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SONET	Synchronous Optical Network
SPT	Shortest Path Tree
SSH	Secure Shell
SSID	Service Set Identifier

Acronym	Description
SSM	Source-Specific Multicast
STP	Spanning Tree Protocol
TACACS+	Terminal Access Controller Access Control System Plus
TBF	Token Bucket Filter
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
ToS	Type of Service
TSS	TCP Maximum Segment Size
Tx	transmit
UDP	User Datagram Protocol
VHD	virtual hard disk
vif	virtual interface
VLAN	virtual LAN
VPC	Amazon virtual private cloud
VPN	virtual private network
VRRP	Virtual Router Redundancy Protocol
WAN	wide area network
WAP	wireless access point
WPA	Wired Protected Access