VYATTA, INC. | Vyatta System

Remote Access API 2.0

REFERENCE GUIDE

Vyatta
Suite 200
1301 Shoreway Road
Belmont, CA 94002
vyatta.com
650 413 7200
1 888 VYATTA 1 (US and Canada)

COPYRIGHT

PROPRIETARY NOTICES

RELEASE DATE: July 2011

DOCUMENT REVISION. R6.3 v01

RELEASED WITH: R6.3.0

PART NO. A0-0237-10-0005

Contents

Quick List of Examples

Use this list to help you locate examples you'd like to look at or try.

Preface

This document describes the various deployment, installation, and upgrade options for Vyatta software.

This preface provides information about using this guide. The following topics are presented:

Intended Audience

This guide is intended for programmers and feature developers intending to deliver a feature or service that is fully integrated with the Vyatta system user interfaces. The intended audience of this guide is experienced system programmers. It is intended for developers and assumes some knowledge of the Vyatta system in general as well as the underlying Vyatta CLI architecture. In addition, the developer should have detailed knowledge of the specific service to be integrated, so that integration can be implemented with appropriate system, process, and data dependencies.

Organization of This Guide

This guide has the following chapters:

Document Conventions

This guide uses the following advisory paragraphs, as follows.

WARNING *Warnings alert you to situations that may pose a threat to personal safety.*

CAUTION *Cautions alert you to situations that might cause harm to your system or damage to equipment, or that may affect service.*

NOTE *Notes provide information you might need to avoid problems or configuration errors.*

This document uses the following typographic conventions.

`Monospace`	Examples, command-line output, and representations of configuration nodes.
`bold Monospace`	Your input: something you type at a command line.
bold	Commands, keywords, and file names, when mentioned inline. Objects in the user interface, such as tabs, buttons, screens, and panes.
italics	An argument or variable where you supply a value.
<key>	A key on your keyboard, such as <Enter>. Combinations of keys are joined by plus signs (“+”), as in <Ctrl>+c.
[key1 | key2]	Enumerated options for completing a syntax. An example is [enable | disable].
num1–numN	A inclusive range of numbers. An example is 1–65535, which means 1 through 65535, inclusive.
arg1..argN	A range of enumerated values. An example is eth0..eth3, which means eth0, eth1, eth2, or eth3.
arg[*arg*...] *arg*[,*arg*...]	A value that can optionally represent a list of elements (a space-separated list and a comma-separated list, respectively).

Vyatta Publications

Full product documentation is provided in the Vyatta technical library. To see what documentation is available for your release, see the *Guide to Vyatta Documentation*. This guide is posted with every release of Vyatta software and provides a great starting point for finding the information you need.

Additional information is available on www.vyatta.com and www.vyatta.org.

Chapter 1: Overview

This chapter provides an overview of the Vyatta system Remote Access API.

This chapter presents the following topics:

Introduction

This feature is available only in the Vyatta Subscription Edition.

This document describes the Vyatta Remote Access API 2.0. This API enables remote command execution on a Vyatta Subscription Edition system over HTTPs using a simple, coherent, 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 Vyatta Remote Access API 1.0.

The API interacts with the Vyatta system 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 using **HTTP GET**, **DELETE**, **PUT**, and **POST** commands, using the format *HTTP-command REST-statement*, as in the following example:

```
GET /rest/conf
```

The specific result of any REST statement depends on the **HTTP** command used to submit it to the remote Vyatta system.

NOTE *The command reference material for REST statements in this guide includes the* ***HTTP*** *commands 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 Vyatta system.*

API Prerequisites

Using the Vyatta Remote Access API 2.0 requires the following:

- A Vyatta Subscription Edition system running Release 6.0 or later software, with HTTPs enabled

- A valid username and password on the Vyatta system
- A system to generate the HTTPs requests to the Vyatta system

Enabling HTTPs on the Vyatta System

To enable HTTPs on the Vyatta system, 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 will be 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, interactions with the Vyatta system consist of an HTTPs request to the system 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 shows a simple request and response.

Description	Command / Response
Request to clear counters on eth0 on 10.0.0.232. The string "dnlhdHRhOnZ5YXR0YQ==" is the bas64 encoding of the username/password pair vyatta:vyatta	POST /rest/op/clear/interfaces/ethernet/eth0/counters Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
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) will suffice.

For example, to generate the request in the previous example using **curl** you would enter the following:

Description	Command
Use **curl** to generate an operational mode request "clear interfaces ethernet eth0 counters" using the username/password pair vyatta:vyatta to the Vyatta system at 10.0.0.232.	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/ethernet/eth0/counters**
System response.	HTTP/1.1 201 Created Transfer-Encoding: chunked Content-Type: application/json Location: rest/op/D462E45FCEFBB5A5 Date: Fri, 19 Feb 2010 21:28:39 GMT Server: lighttpd/1.4.19

For examples of Perl scripts that use the API, see the following files on the Vyatta system:

```
/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 eth0 to 10.0.0.231/24 using the CLI you would enter the following command in configuration mode.

```
set interfaces ethernet eth0 address 10.0.0.231/24
```

Within the API, however, the last slash character ("/") character would be 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 string must be represented as follows.

```
rest/conf/set/interfaces/ethernet/eth0/address/10.0.0.231%2F24
```

The following table shows the encodings required for the various special characters:

Example 1-1 Special Characters Encodings

Special Character	Encoding
Tab	%09
Space	%20
"	%22
#	%23
%	%25
&	%26
(	%28
)	%29
+	%2B
,	%2C
.	%2E

Example 1-1 Special Characters Encodings

Special Character	Encoding
/	%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 succeeded by ensuring that the HTTP status code is a 200-level (i.e., 2xx “Ok”) response code. The Content-Type information returned in the response header should also be checked to verify that it is correct.

There are three types of response bodies that can be returned in response to a request:

- An empty body
- A JSON-formatted body
- A plain text body

The body of the response returned, and its format, depends on the requested HTTP Content-Type and the command 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 containing operational mode command output (for example, output from the **show version** command) have plain text bodies.

JSON responses vary with the command issued. Additional command status codes can also be returned in the body of the response. Example 1-2 shows a JSON-formatted response body—in this case, the currently active operational mode processes.

Example 1-2 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 either hash entries, arrays, or nested combinations of these. In the response body shown above:

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

Processing JSON-formatted responses is well-supported using support libraries available in most programming languages. Further information can be found at http://www.json.org.

Sample Workflows

Most things you can do with the Vyatta system you can do through the API. The examples in this section show the workflow for some common or useful operations using the API by means of a scripting or compiled language of choice.

Multiple system configuration deployments

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

Monitoring an Interface across Systems

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

Chapter 2: Configuration Mode

This chapter provides a description of the functionality available in configuration mode of the Vyatta Remote Access API.

This chapter presents the following topics:

Overview

The configuration mode of the Vyatta Remote Access API allows you to update the system configuration of the remote Vyatta system. The workflow for configuration using the API is the same as using the system directly: begin a configuration session, make configuration changes, commit changes, optionally view them and save them. Configuration mode is accessed 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 complete to release the resources associated with the session.

Summary of Tasks

You can perform the following tasks using the commands in this chapter.

Start a configuration session and enter configuration mode	POST /rest/conf CLI equivalent: configure (in operational mode)
Create or modify configuration information	PUT /rest/conf/<conf-id>/set/<path> CLI equivalent: set
Delete configuration information	PUT rest/conf/<conf-id>/delete/<path> CLI equivalent: delete
Issue a configuration action command, such as commit, save, load, merge, or show	POST /rest/conf/<conf-id>/<cmd> CLI equivalent: commit, save, load, merge, show.
Display configuration information	GET /rest/conf/<conf-id>/<path> CLI equivalent: show (in configuration mode)
List active configuration mode sessions	GET /rest/conf CLI equivalent: None.
Exit the specified configuration session	DELETE /rest/conf/<conf-id> CLI equivalent: exit

Configuration Mode Session Example

The following is an example of a Remote Access API request to a Vyatta system at 10.0.0.232 to initiate a configuration mode session.

Example 2-1 Generating a configuration mode session request

Description	Command/Response

Request a new configuration mode session request. This is the equivalent of the CLI **configure** command.	`POST /rest/conf`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

The HTTPs server located on port 443 will receive this request and identify 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. Once validated, the request will run using the credentials of the specified user (in this case "vyatta").

At this point the action and the remaining elements in the header are used to 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 Vyatta system to create a new configuration session for this request and return a unique ID for this configuration session.

The response generated by the remote system is as shown below.

Example 2-2 System response to configuration mode session request

Description	Command/Response

System response.	`HTTP/1.1 201 Created`
	`Transfer-Encoding: chunked`
	`Content-Type: application/json`
	`Location: rest/conf/2957FC9CA1FE3B3C`
	`Date: Wed, 05 May 2010 18:52:48 GMT`
	`Server: lighttpd/1.4.19`

The 201 response code indicates that the request succeeded and that remote system was able to create a new configuration tree associated with the ID "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.*

In the following example:

- 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.

Example 2-3 A configuration mode session

Description	Command/Response

Request to view active configuration sessions. The only difference between this and the previous example is that the **POST** command is replaced with the **GET** command.	`GET /rest/conf`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 200 response code indicates that the request succeeded. In this case the output (in JSON format) shows that there are two active configuration sessions associated with the user that initiated the command (user "vyatta") and each 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 Jan 1, 1970). The "modified" field indicates whether or not there are uncommitted changes in the configuration session.	`HTTP/1.1 200 OK`
	`Content-Type: application/json`
	`Content-Length: 412`
	`Date: Thu, 06 May 2010 03:15:18 GMT`
	`Server: lighttpd/1.4.19`
	
	`{`
	`  "session": [`
	`    {`
	`      "id": "2957FC9CA1FE3B3C",`
	`      "username": "vyatta",`
	`      "description": "",`
	`      "started": "1273085568",`
	`      "modified": "false",`
	`      "updated": "1273085568"`
	`    },`
	`    {`
	`      "id": "F84307222F469A8A",`
	`      "username": "vyatta",`
	`      "description": "",`
	`      "started": "1273085337",`
	`      "modified": "false",`
	`      "updated": "1273085337"`
	`    }`
	`  ],`
	`  "message": " "`
	`}`

Example 2-3 A configuration mode session

Description	Command/Response
Request to change the ssh port to 1011. This is equivalent to the CLI configuration mode command **set service ssh port 1011**. Note that the configuration session ID that was created above is used in the URI.	PUT /rest/conf/2957FC9CA1FE3B3C/set/service/ssh/port/1011 Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==
System response. The 200 response code indicates that the request succeeded.	HTTP/1.1 200 OK Transfer-Encoding: chunked Content-Type: application/json Date: Thu, 06 May 2010 03:48:46 GMT Server: lighttpd/1.4.19
Request to commit the configuration change. This is equivalent to the CLI configuration mode command **commit**.	POST /rest/conf/2957FC9CA1FE3B3C/commit Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==

System response. The 200 response code indicates that the request succeeded. Notice that the SSH daemon (sshd) was restarted due to the configuration change.	HTTP/1.1 200 OK
	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 "
	}

Request to save the active configuration. This is equivalent to the CLI configuration mode command **save**.	POST /rest/conf/2957FC9CA1FE3B3C/save Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==

Example 2-3 A configuration mode session

Description	Command/Response

System response. The 200 response code indicates that the request succeeded. The message shows where the file was saved.	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 configuration to '/config/config.boot'...\n Done\n "
	}

Request to end the configuration session. This is equivalent to the CLI configuration mode command **exit**.	DELETE /rest/conf/2957FC9CA1FE3B3C Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==

System response. The 200 response code indicates that the request succeeded.	HTTP/1.1 200 OK
	Content-Type: application/json
	Content-Length: 21
	Date: Thu, 06 May 2010 03:59:12 GMT
	Server: lighttpd/1.4.19
	
	{
	"message": " "
	}

Request to view active configuration sessions.	GET /rest/conf Host: 10.0.0.232 Accept: application/json Vyatta-Specification-Version: 0.1 Authorization: Basic dnlhdHRhOnZ5YXR0YQ==

Example 2-3 A configuration mode session

Description	Command/Response

System response. The 200 response code indicates that the request succeeded. In this case, the session that ended no longer shows up in the list of active configuration sessions. A single active configuration session remains.	HTTP/1.1 200 OK
	Content-Type: application/json
	Content-Length: 226
	Date: Thu, 06 May 2010 03:59:30 GMT
	Server: lighttpd/1.4.19
	
	{
	"session": [
	{
	"id": "F84307222F469A8A",
	"username": "vyatta",
	"description": "",
	"started": "1273085337",
	"modified": "false",
	"updated": "1273085337"
	}
	],
	"message": " "
	}

Configuration Mode Command Reference

This section presents the following commands:

DELETE /rest/conf/<conf-id>	Ends the specified configuration mode session. CLI equivalent: exit
GET /rest/conf	Retrieves a list of the active configuration sessions for the host at the specified IP address. CLI equivalent: None.
GET /rest/conf/<conf-id>/<path>	Returns the definition of a configuration mode parameter, its current value, and a list of its child parameters. CLI equivalent: show (in configuration mode)
POST /rest/conf	Creates a new configuration session ID that provides access to configuration mode commands. CLI equivalent: configure (in operational mode)
POST /rest/conf/<conf-id>/<cmd>	Issues a configuration mode action command. CLI equivalent: commit, save, load, merge, show.
PUT rest/conf/<conf-id>/delete/<path>	Deletes configuration information. CLI equivalent: delete
PUT /rest/conf/<conf-id>/set/<path>	Set configuration information. CLI equivalent: set

DELETE /rest/conf/<conf-id>

Ends the specified configuration mode session.

CLI equivalent: **exit**

Synopsis

DELETE /rest/conf/*conf-id*

Mode

Configuration mode.

Parameters

conf-id	The ID of the configuration session to be ended.

Request/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 end and exit a configuration mode session, removing the session ID and releasing system resources associated with the session.

Example

Example 2-4 Ending a configuration session.

Description	Command/Response

End 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==`

System response.	`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

Retrieves a list of the active configuration sessions for the host at the specified IP address.

CLI equivalent: None.

Synopsis

GET /rest/conf

Mode

Configuration mode.

Parameters

None.

Default

None.

Request/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 retrieve a list of the active configuration sessions for a host.

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

Example

Example 2-5 Retrieve a list of active configuration sessions.

Description	Command/Response

Request a list of all active configuration sessions on 10.0.0.232.	`GET /rest/conf`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. Note the session ID returned in the **Location:** field.	`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"`
	`},`
	<cont>

Example 2-5 Retrieve a list of active configuration sessions.

Description	Command/Response

	<pre>{
	"id": "DE54D909FA4047B2",
	"username": "vyatta",
	"started": "1266614404",
	"modified": "false",
	"updated": "1266614404",
	"description": "current-configuration"
	}
	]
	}</pre>

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

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

CLI equivalent: **show** (in configuration mode)

Synopsis

GET /rest/conf/*conf-id/path*

Mode

Configuration mode.

Parameters

conf-id	The configuration session ID associated with the configuration session.
path	The path to the configuration node.

Default

None.

Request/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 obtain configuration information from the Vyatta system.

As in the CLI, the user must create a configuration session before parameters may 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 is as follows:

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

State: Can be [active, set, delete, none]. Denotes if this node is active in the current configuration, not-active, 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: If this node is last element of the tree.

Mandatory: If this is required in the configuration.

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

Default: If this value node has a default value this will be returned.

Help: Help text.

Val_help: If 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 along with their current state in the local and active configuration tree.

Example

Example 2-6 Retrieve configuration information for "service ssh" node

Description	Command/Response

Request 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==`

Example 2-6 Retrieve configuration information for “service ssh” node

System response.

```
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.

CLI equivalent: **configure** (in operational mode)

Synopsis

POST /rest/conf[/*description*]

Mode

Configuration mode.

Parameters

description	Optional. A descriptive tag for the session.

Default

None.

Request/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.

Example

Example 2-7 Create a new configuration session

Description	Command/Response

Request to create a configuration session on 10.0.0.232.	`POST /rest/conf`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

The response. Note the session ID returned in the **Location:** field.	`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`

Example 2-8 Create a new configuration session using a description

Description	Command/Response

Request to create a configuration session on 10.0.0.232 using the string "XYZ" as a description string.	`POST /rest/conf/XYZ`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

The response. Again the session ID is returned in the **Location:** field.	`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`

Example 2-8 Create a new configuration session using a description

Request a list of all active configuration sessions on 10.0.0.232.	`GET /rest/conf`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System 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.	`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"`
	`  }`
	`  ]`
	`}`

For example, end 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==`

System response.	`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>

Issues a configuration mode action command.

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

Synopsis

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

Mode

Configuration mode.

Parameters

conf-id	The configuration session ID associated with the configuration session.
cmd	The configuration action command to execute.
path	Optional. The path to the configuration node to be affected by the command. Equivalent to the CLI configuration mode parameters.

Default

None.

Request/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 issue a configuration action command. Configuration action commands are configuration commands that do not modify local configuration nodes in the way that **set**, **delete**, and **run** do, for example. Supported commands are **commit**, **save**, **load**, **discard**, **merge**, and **show**.

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 is the equivalent of entering configuration mode in the CLI. Also, as in the CLI, changes must be committed for them to be applied to the system.

Example

Example 2-9 Commiting changes.

Description	Command/Response

Request to **commit** changes on 10.0.0.232.	`POST /rest/conf/27755B4BC823272E/commit`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. In this case no changes were made so none were committed.	`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.

CLI equivalent: **delete**

Synopsis

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

Mode

Configuration mode.

Parameters

conf-id	The configuration session ID associated with the configuration session.
path	The path to the configuration node. Equivalent to the CLI configuration mode parameters.

Default

None.

Request/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. Using this command is the equivalent to using 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 escaped within the URI string. The API automatically converts the escaped characters back to the intended character.

NOTE *Characters that are not valid url characters need to be url encoded (for example, spaces). These values will be converted back on the Vyatta system.*

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

Example

Example 2-10 Issue command to delete the SSH service.

Description	Command/Response

Request to delete 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==`

System response.	`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>

Set configuration information.

CLI equivalent: **set**

Synopsis

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

Mode

Configuration mode.

Parameters

conf-id	The configuration session ID associated with the configuration session.
path	The path to the configuration node. Equivalent to the CLI configuration mode parameters.

Default

None.

Request/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. Using this command is the equivalent to using 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 escaped within the URI string. The API automatically converts the escaped characters back to the intended character.

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

Example

Example 2-11 Change the SSH port.

Description	Command/Response

Request to change 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==`

System response.	`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 3: Operational Mode

This chapter provides a description of the functionality available in operational mode of the Vyatta Remote Access API.

This chapter presents the following topics:

Overview

Operational mode provides the ability to remotely execute operational mode commands and view their output. Each command executed initiates a process which may have a finite lifespan (for example, **show version**) or may run indefinitely until explicitly stopped (for example, **ping** *address*).

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

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

NOTE *Operational mode commands that are interactive in nature (e.g., 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).

Summary of Tasks

You can perform the following tasks using the commands in this chapter.

Execute an operational mode command	POST /rest/op/<path>
Retrieve operational mode parameter definitions	GET /rest/op/<path>
View a list of operational mode processes	GET /rest/op
Retrieve output	GET /rest/op/<process-id>
End an operational mode process	DELETE /rest/op/<process-id>

Single-Output Example

Example 3-1 shows a request to a Vyatta system at 10.0.0.232 to view the system version information. It is equivalent to the operational mode command **show version**. **This command has a finite lifespan and the system resources it uses are freed once its output is read.**

Example 3-1 Generating a request to view version information

Description	Command/Response

Request to view system version information. Note that **rest/op** is used rather than **rest/conf**. Also note that we did not begin a "session" like we did in configuration mode.	`POST /rest/op/show/version`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 201 response code indicates that the request succeeded and that the command output is available at the specified **Location:**. To view the output a separate request must be submitted specifying the output location.	`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`

Request to view **show version** command output. Note that the **GET** command is used and the **Location:** information from above is used in the URI.	`GET /rest/op/137AA3B22A362CA3`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 200 response code indicates that the request succeeded. The output to the **show version** command is returned in plain text format.	`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.larkspurse.04280031`
	`Description: 999.larkspurse.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`

Request to view **show version** command output a second time. This time the result will be different.	`GET /rest/op/137AA3B22A362CA3`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

Example 3-1 Generating a request to view version information

System response. The 410 response code indicates that the request failed and the the output is “gone” as it was “consumed” by the first request. All system resources used by the initial command have been freed.	`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`

Continuous-Output Example

The previous example showed the execution of a command that generated its output and finished (finite lifespan). In this example we look at a command that continues to generate output until it is explicitly stopped (infinite lifespan). The client may need to make several requests for data before the 410 response is received (if the process generating the output ends). Each response is a sequential piece of output from the command that was executed. If the process that is generating the output does not end, output can be retrieved only while the process is running. Once the process is stopped the output will be deleted.

The following is an example of a request to a Vyatta system at 10.0.0.232 to ping another device. It is equivalent to the operational mode command **ping address**.

Example 3-2 Generating a request to ping a device

Description	**Command/Response**

Request to ping the device at 10.0.0.1 from 10.0.0.232.	`POST /rest/op/ping/10.0.0.1`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 201 response code indicates that the request succeeded and that the command output is available at the specified **Location:**. To view the output a separate request must be submitted specifying the output location.	`HTTP/1.1 201 Created`
	`Transfer-Encoding: chunked`
	`Content-Type: application/json`
	`Location: rest/op/02B3479CA1522F2A`
	`Date: Fri, 07 May 2010 17:10:30 GMT`
	`Server: lighttpd/1.4.19`

Request to view **ping** command output. Note that the **GET** command is used and the **Location:** information from above is used in the URI.	`GET /rest/op/02B3479CA1522F2A`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

Example 3-2 Generating a request to ping a device

System response. The 200 response code indicates that the request succeeded. The output to the **ping** command is returned in plain text format. The **ping** command continues to run in the background and generate output.	HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 1164 Date: Fri, 07 May 2010 17:10:49 GMT Server: lighttpd/1.4.19 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: 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

Example 3-2 Generating a request to ping a device

Request to view **ping** command output a second time. This time the result will be different.	`GET /rest/op/02B3479CA1522F2A`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

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 will continue to run in the background until we terminate it.	`HTTP/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 248`
	`Date: Fri, 07 May 2010 17:10:52 GMT`
	`Server: lighttpd/1.4.19`
	
	`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`

Request to view the list of active operational mode processes.	`GET /rest/op`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 200 response code indicates that the request succeeded. The output (in JSON format) displays the requested information showing a single active operational mode process.	`HTTP/1.1 200 OK`
	`Content-Type: application/json`
	`Content-Length: 193`
	`Date: Fri, 07 May 2010 17:10:58 GMT`
	`Server: lighttpd/1.4.19`
	
	`{`
	`  "process": [`
	`    {`
	`      "username": "vyatta",`
	`      "started": "1273252231",`
	`      "updated": "8",`
	`      "id": "02B3479CA1522F2A",`
	`      "command": "ping 10.0.0.1"`
	`    }`
	`  ]`
	`}`

Example 3-2 Generating a request to ping a device

Request to stop an active operational mode process. Note that the **DELETE** command is used and the **Location:** information (containing the unique operational mode process ID) from above is used in the URI. This will stop the **ping** command and remove any remaining output, freeing up all resources used by the command.	`DELETE /rest/op/02B3479CA1522F2A`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 200 response code indicates that the request succeeded.	`HTTP/1.1 200 OK`
	`Content-Type: application/json`
	`Content-Length: 21`
	`Date: Fri, 07 May 2010 17:11:20 GMT`
	`Server: lighttpd/1.4.19`
	
	`{`
	`  "message": " "`
	`}`

Request to view the list of active operational mode processes.	`GET /rest/op`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response. The 200 response code indicates that the request succeeded but we see that there are no active operational mode processes as the response body is empty.	`HTTP/1.1 200 OK`
	`Content-Type: application/json`
	`Content-Length: 0`
	`Date: Fri, 07 May 2010 17:39:46 GMT`
	`Server: lighttpd/1.4.19`

Operational Mode Command Reference

This section presents the following commands.

DELETE /rest/op/<process-id>	Ends an operational mode process.
GET /rest/op	Returns a list of the operational mode commands that are still running and/or still have data output pending.
GET /rest/op/<path>	Retrieves the definition of an operational mode command and a list of its children.
GET /rest/op/<process-id>	Returns the output from an operational mode command.
POST /rest/op/<path>	Issues an operational mode command.

DELETE /rest/op/<process-id>

Ends an operational mode process.

Synopsis

DELETE /rest/op/*process-id*

Mode

Operational mode.

Parameters

process-id	The process ID to end.

Default

None.

Request/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 (kill) an operational mode process and release the system resources associated with it.

Example

Example 3-3 Terminating an operational mode process

Description	Command / Response

Request 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==`

System response.	`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

Returns a list of the operational mode commands that are still running and/or still have data output pending.

Synopsis

GET /rest/op

Mode

Operational mode.

Parameters

None.

Default

None.

Request/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 retrieve a list of the operational mode commands that are still running and/or still have data output pending.

Processes 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 initiated by the user as specified in the authentication header will be returned. Time values (start and update) are returned in UNIX epoch time (seconds since January 1st, 1970).

Example

Example 3-4 Listing active operational mode commands

Description	Command / Response

Request 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==`

System response.	`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"`
	`    },`
	<cont>

Example 3-4 Listing active operational mode commands

Response (cont'd).

```
 {
      "username": "vyatta",
      "started": "1266615495",
      "updated": "1919251319",
      "id": "A12A9F9707621658",
      "command": "show interfaces ethernet
eth0"
    },
    {
      "username": "vyatta",
      "started": "1266614874",
      "updated": "1919251319",
      "id": "D90078870BEB3FF5",
      "command": "show users"
    }
  ]
}
```

GET /rest/op/<path>

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

Synopsis

GET /rest/op/*path*

Mode

Operational mode.

Parameters

path	The path to the operational command.

Default

None.

Request/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 retrieve the definition of an operational mode command and a list of its children.

Data is returned in a JSON hash. The response returns help for the the node being requested, a Boolean value indicating whether this command can be executed, and a list of the node's children and enumered values, if available.

Example

Example 3-5 Retrieving operational mode parameters

Description	Command / Response

Request to get operational mode parameter definitions for “ethernet interfaces eth0” on 10.0.0.232.	`GET /rest/op/show/interfaces/ethernet/eth0`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response.	`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": [`
	`"eth0",`
	`"eth1",`
	`"eth2"`
	`],`
	`"action": "true",`
	`"help": " Show specified ethernet interface information"`
	`}`

GET /rest/op/<process-id>

Returns the output from an operational mode command.

Synopsis

GET /rest/op/*process-id*

Mode

Operational mode.

Parameters

process-id	The process ID used to retrieve operational mode command output.

Default

None.

Request/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 retrieve the output from an operational command.

Note that some commands may not terminate by themselves. It is the responsibility of the developer to manage non-terminating processes. Also note that queries can sometimes be generated faster than the command can produce data, in this case the command will return with 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 return with a 410 response do not require any further process management or client-initiated process deletion.

The response is not JSON, but plain text.

Example

Example 3-6 Retrieving operational command output

Description	Command / Response

Request to retrieve the output from the specified process ID on 10.0.0.232.	`GET /rest/op/A12A9F9707621658`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

System response.	`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`
	
	`eth0: <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 eth0`
	`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/op/<path>

Issues an operational mode command.

Synopsis

POST /rest/op/*path*

Mode

Operational mode.

Parameters

path	Uniquely identifies the location in the operational mode command tree.

Default

None.

Request/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 issue an operational mode CLI command on the remote system.

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

Note that a success (2xx) response 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, as well as command success or failure.

The **Location:** parameter identifies the resource location. A further operational mode **GET** to this location will return any results associated with the command. However, as the command may still be running, successive **GET** commands may be required to obtain the full response (see the section *"Continuous-Output Example" on page 37* section for additional details).

Example

Example 3-7 Clearing counters

Description	Command / Response

Request to clear counters on "ethernet interfaces eth0" on 10.0.0.232.	`POST /rest/op/clear/interfaces/ethernet/eth0/counters`
	`Host: 10.0.0.232`
	`Accept: application/json`
	`Vyatta-Specification-Version: 0.1`
	`Authorization: Basic dnlhdHRhOnZ5YXR0YQ==`

The response. Note the process ID returned in the **Location:** field.	`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`

Appendix A: HTTP Status Codes

This appendix lists the HTTP status codes returned by the Vyatta system.

Table A-1 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	Data is returned
	202 Accepted	Process is still running but there is no new data
	404 Not Found	Failure to retrieve Operational mode configuration data
	410 Gone	End of data retrieval for the process
	500 Internal Server Error	Server request error
• POST		
	201 Created	Initiating a process
	400 Bad Request	Client request error
	500 Internal Server Error	Server side error
Configuration Mode		
	400 Bad Request	Client request error (malformed request)
• DELETE		

Table A-1 HTTP Status Codes

Command	Status Code	Description
	400 Bad Request	Client request error
• GET		
	404 Not Found	Configuration data not found
	500 Internal Server Error	Server side error
• POST		
	201 Created	Configuration resource created
	400 Bad Request	Client request error
	500 Internal Server Error	Server side error
• PUT		
	400 Bad Request	Client request error

Glossary of Acronyms

ACL	access control list
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
AS	autonomous system
ARP	Address Resolution Protocol
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
DMZ	demilitarized zone
DN	distinguished name
DNS	Domain Name System

DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
eBGP	external BGP
EGP	Exterior Gateway Protocol
ECMP	equal-cost multipath
ESP	Encapsulating Security Payload
FIB	Forwarding Information Base
FTP	File Transfer Protocol
GRE	Generic Routing Encapsulation
HDLC	High-Level Data Link Control
I/O	Input/Ouput
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
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
ISP	Internet Service Provider
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
MIB	Management Information Base
MLPPP	multilink PPP
MRRU	maximum received reconstructed unit
MTU	maximum transmission unit
NAT	Network Address Translation
ND	Neighbor Discovery
NIC	network interface card
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
PKI	Public Key Infrastructure
PPP	Point-to-Point Protocol
PPPoA	PPP over ATM
PPPoE	PPP over Ethernet
PPTP	Point-to-Point Tunneling Protocol
PVC	permanent virtual circuit
QoS	quality of service
RADIUS	Remote Authentication Dial-In User Service

RIB	Routing Information Base
RIP	Routing Information Protocol
RIPng	RIP next generation
Rx	receive
SLAAC	Stateless Address Auto-Configuration
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SONET	Synchronous Optical Network
SSH	Secure Shell
SSID	Service Set Identifier
STP	Spanning Tree Protocol
TACACS+	Terminal Access Controller Access Control System Plus
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
ToS	Type of Service
Tx	transmit
UDP	User Datagram Protocol
vif	virtual interface
VLAN	virtual LAN
VPN	Virtual Private Network
VRRP	Virtual Router Redundancy Protocol
WAN	wide area network
WAP	wireless access point
WPA	Wired Protected Access