



Fabric Engine and VOSS 9.1 RESTCONF API Guide

Implementation, Configuration, and Management for
VOSS and Fabric Engine

9039036-00 Rev AA
October 2024



Copyright © 2024 Extreme Networks, Inc.

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, see: <https://www.extremenetworks.com/about-extreme-networks/company/legal/trademarks>

Open Source Declarations

Some software files have been licensed under certain open source or third-party licenses.

End-user license agreements and open source declarations can be found at: <https://www.extremenetworks.com/support/policies/open-source-declaration/>



Table of Contents

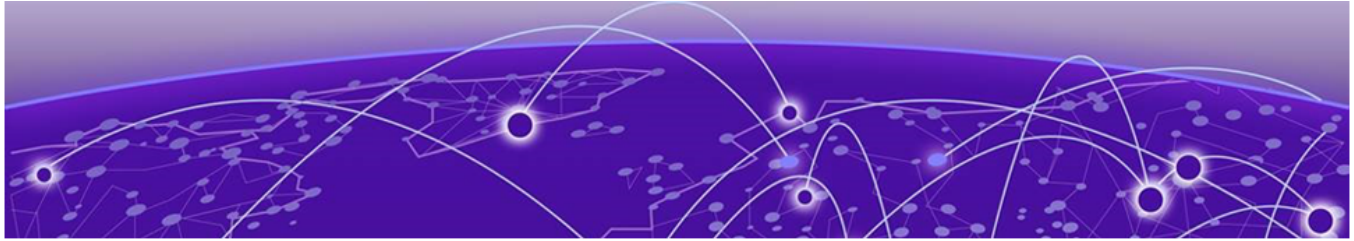
Abstract.....	5
Preface.....	6
Purpose.....	6
Conventions.....	6
Text Conventions.....	7
Documentation and Training.....	9
Open Source Declarations.....	9
Training.....	9
Help and Support.....	9
Subscribe to Product Announcements.....	10
Send Feedback.....	10
RESTCONF APIs.....	12
RESTCONF Protocol.....	13
YANG Data Models.....	15
JSON Representation.....	16
CRUD Operations.....	19
Query Parameters.....	20
depth.....	21
content.....	28
insert.....	33
point.....	36
formatted.....	38
filter.....	39
Error Response Codes.....	41
Log in to the RESTCONF Server.....	42
RESTCONF Root Resource.....	42
Authentication.....	43
Access the RERESTCONFSTCONF Datastores.....	43
RESTCONF API and CLI Commands Examples.....	47
Port CLI Commands.....	47
RESTCONF API for Port CLI Commands.....	48
PATCH.....	48
GET.....	49
MLT CLI Commands.....	49
RESTCONF API for MLT CLI Command.....	49
POST or PATCH.....	49
GET.....	50

VLAN CLI Commands.....	50
RESTCONF API for VLAN CLI Command.....	50
POST or PATCH.....	50
GET.....	51
DELETE.....	51
VRF CLI Commands.....	51
RESTCONF API for VRF CLI Commands.....	52
POST.....	52
GET.....	56
DELETE.....	57
I-SID CLI Commands.....	58
RESTCONF API for I-SID CLI Command.....	59
POST.....	59
PATCH.....	61
GET.....	63
DELETE.....	63



Abstract

This API Guide for Fabric Engine and VOSS version 9.1 provides instructions on configuring and managing ExtremeSwitching and Virtual Services Platform (VSP) devices using the RESTCONF protocol with YANG data models. It explains RESTCONF APIs, detailing CRUD operations (GET, POST, PATCH, DELETE), supported query parameters (such as depth, content, filter), and JSON data representation. The guide includes procedures for logging into the RESTCONF server, accessing datastores, and performing network automation tasks. It also offers specific examples of API requests and responses, covering various features such as VLAN, port configurations, and VRF commands.



Preface

[Purpose](#) on page 6
[Conventions](#) on page 6
[Documentation and Training](#) on page 9
[Help and Support](#) on page 9
[Send Feedback](#) on page 10

Read the following topics to learn about:

- The meanings of text formats used in this document.
- Where you can find additional information and help.
- How to reach us with questions and comments.

Purpose

This document provides information to use Representational State Transfer Configuration Protocol (RESTCONF) that runs on the following product families:

- ExtremeSwitching 5320 Series
- ExtremeSwitching 5420 Series
- ExtremeSwitching 5520 Series
- ExtremeSwitching 5720 Series
- ExtremeSwitching 7520 Series
- ExtremeSwitching 7720 Series
- ExtremeSwitching VSP 4900 Series
- ExtremeSwitching VSP 7400 Series

Conventions

To help you better understand the information presented in this guide, the following topics describe the formatting conventions used for notes, text, and other elements.

Text Conventions

The following tables list text conventions that can be used throughout this document.

Table 1: Notes and warnings

Icon	Notice type	Alerts you to...
	Tip	Helpful tips and notices for using the product.
	Note	Useful information or instructions.
	Important	Important features or instructions.
	Caution	Risk of personal injury, system damage, or loss of data.
	Warning	Risk of severe personal injury.

Table 2: Text conventions

Convention	Description
The words <i>enter</i> and <i>type</i>	When you see the word <i>enter</i> in this guide, you must type something, and then press the Return or Enter key. Do not press the Return or Enter key when an instruction simply says <i>type</i> .
Key names	Key names are written in boldface, for example Ctrl or Esc . If you must press two or more keys simultaneously, the key names are linked with a plus sign (+). Example: Press Ctrl+Alt+Del
NEW!	New information. In a PDF, this is searchable text.

Table 3: Command syntax

Convention	Description
Angle brackets (< >)	Angle brackets (< >) indicate that you choose the text to enter based on the description inside the brackets. Do not type the brackets when you enter the command.

Table 3: Command syntax (continued)

Convention	Description
	If the command syntax is <code>cfm maintenance-domain maintenance-level <0-7></code> , you can enter <code>cfm maintenance-domain maintenance-level 4</code> .
Bold text	Bold text indicates the GUI object name you must act upon. Examples: - Select OK. - On the Tools menu, choose Options.
Braces ({ })	Braces ({ }) indicate required elements in syntax descriptions. Do not type the braces when you enter the command. For example, if the command syntax is <code>ip address {A.B.C.D}</code> , you must enter the IP address in dotted, decimal notation.
Brackets ([])	Brackets ([]) indicate optional elements in syntax descriptions. Do not type the brackets when you enter the command. For example, if the command syntax is <code>show clock [detail]</code> , you can enter either <code>show clock</code> or <code>show clock detail</code> .
Ellipses (...)	An ellipsis (...) indicates that you repeat the last element of the command as needed. For example, if the command syntax is <code>ethernet/2/1 [<parameter> <value>] ...</code> , you enter <code>ethernet/2/1</code> and as many parameter-value pairs as you need.
<i>Italic Text</i>	Italics emphasize a point or denote new terms at the place where they are defined in the text. Italics are also used when referring to publication titles that are not active links.
Plain Courier Text	Plain Courier text indicates command names, options, and text that you must enter. Plain Courier text also indicates command syntax and system output, for example, prompts and system messages. Examples: <code>show ip route</code> <code>Error: Invalid command syntax [Failed][2013-03-22 13:37:03.303 -04:00]</code>

Table 3: Command syntax (continued)

Convention	Description
Separator (>)	A greater than sign (>) shows separation in menu paths. For example, in the Navigation pane, expand Configuration > Edit .
Vertical Line ()	A vertical line () separates choices for command keywords and arguments. Enter only one choice. Do not type the vertical line when you enter the command. For example, if the command syntax is <code>access-policy by-mac action { allow deny }</code> , you enter either <code>access-policy by-mac action allow</code> or <code>access-policy by-mac action deny</code> , but not both.

Documentation and Training

Find Extreme Networks product information at the following locations:

[Current Product Documentation](#)

[Release Notes](#)

[Hardware and Software Compatibility](#) for Extreme Networks products

[Extreme Optics Compatibility](#)

[Other Resources](#) such as articles, white papers, and case studies

Open Source Declarations

Some software files have been licensed under certain open source licenses. Information is available on the [Open Source Declaration](#) page.

Training

Extreme Networks offers product training courses, both online and in person, as well as specialized certifications. For details, visit the [Extreme Networks Training](#) page.

Help and Support

If you require assistance, contact Extreme Networks using one of the following methods:

[Extreme Portal](#)

Search the GTAC (Global Technical Assistance Center) knowledge base; manage support cases and service contracts; download software; and obtain product licensing, training, and certifications.

[The Hub](#)

A forum for Extreme Networks customers to connect with one another, answer questions, and share ideas and feedback. This community is monitored by Extreme Networks employees, but is not intended to replace specific guidance from GTAC.

Call GTAC

For immediate support: (800) 998 2408 (toll-free in U.S. and Canada) or 1 (408) 579 2800. For the support phone number in your country, visit www.extremenetworks.com/support/contact.

Before contacting Extreme Networks for technical support, have the following information ready:

- Your Extreme Networks service contract number, or serial numbers for all involved Extreme Networks products
- A description of the failure
- A description of any actions already taken to resolve the problem
- A description of your network environment (such as layout, cable type, other relevant environmental information)
- Network load at the time of trouble (if known)
- The device history (for example, if you have returned the device before, or if this is a recurring problem)
- Any related RMA (Return Material Authorization) numbers

Subscribe to Product Announcements

You can subscribe to email notifications for product and software release announcements, Field Notices, and Vulnerability Notices.

1. Go to [The Hub](#).
2. In the list of categories, expand the **Product Announcements** list.
3. Select a product for which you would like to receive notifications.
4. Select **Subscribe**.
5. To select additional products, return to the **Product Announcements** list and repeat steps 3 and 4.

You can modify your product selections or unsubscribe at any time.

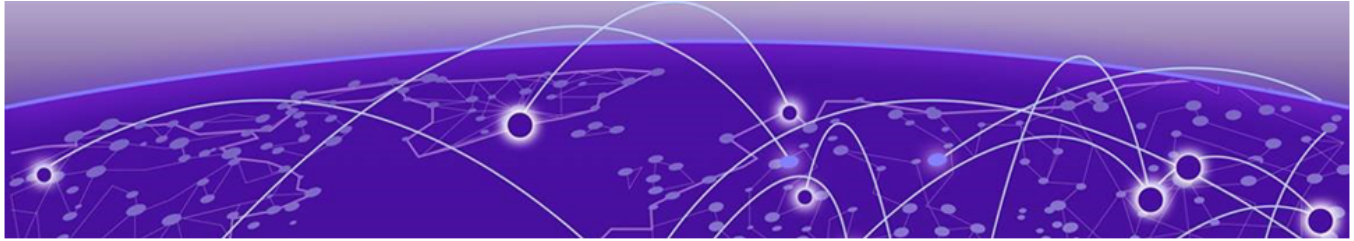
Send Feedback

The User Enablement team at Extreme Networks has made every effort to ensure that this document is accurate, complete, and easy to use. We strive to improve our documentation to help you in your work, so we want to hear from you. We welcome all feedback, but we especially want to know about:

- Content errors, or confusing or conflicting information.
- Improvements that would help you find relevant information.
- Broken links or usability issues.

To send feedback, email us at documentation@extremenetworks.com.

Provide as much detail as possible including the publication title, topic heading, and page number (if applicable), along with your comments and suggestions for improvement.



RESTCONF APIs

You can access the RESTCONF API documentation on your switch using the following URL:

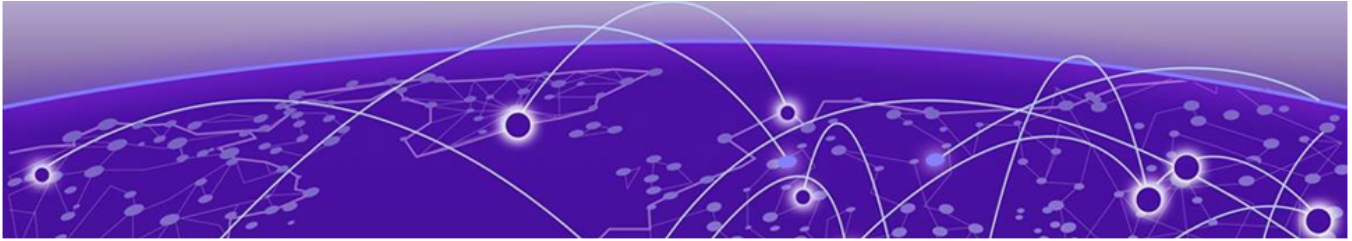
`http(s)://<IP>:<tcp-port>/apps/restconfdoc/`

Replace <IP> with the management IP address of your switch and <tcp-port> with the TCP port configured for RESTCONF. For example, `http://192.0.2.16:8080/apps/restconfdoc/`.

The on-switch URL works only if you enable the RESTCONF feature on the switch.

For more information about configuring RESTCONF on the switch, see the *VOSS User Guide* or the *Fabric Engine User Guide*.

You can also access the RESTCONF API documentation online through the Developer Center (<https://www.extremenetworks.com/support/documentation-api/>).



RESTCONF Protocol

Representational State Transfer Configuration Protocol (RESTCONF) is a standard protocol based on HTTP or HTTPS that provides a programmatic interface to access data defined in YANG, using the datastore concepts defined in the Network Configuration Protocol (NETCONF). YANG is a data modeling language that together with RESTCONF, provides the tools that network administrators need to automate configuration tasks across heterogeneous devices in a software-defined network.

The RESTCONF interface allows client applications to access and manipulate configuration data, state data, data-model-specific Remote Procedure Call (RPC) operations, and event notifications on a networking device, in a modular and extensible manner.

The API uses common HTTP operations, such as **GET**, **POST**, **PATCH**, and **DELETE**, on a conceptual datastore containing YANG-defined data. Request and response messages can be in JSON format. The YANG data model explicitly and precisely determines the structure, syntax, and semantics of the data. The YANG modules are vendor-neutral and include models that are part of the [OpenConfig](#) project, standard [IETF](#) models, as well as some native models.

This model-driven approach to network device programmability allows API predictability, use of familiar HTTP tools and programming libraries, and ease of integration for the large pool of developers or engineers accustomed to a [REST](#)-like interface.

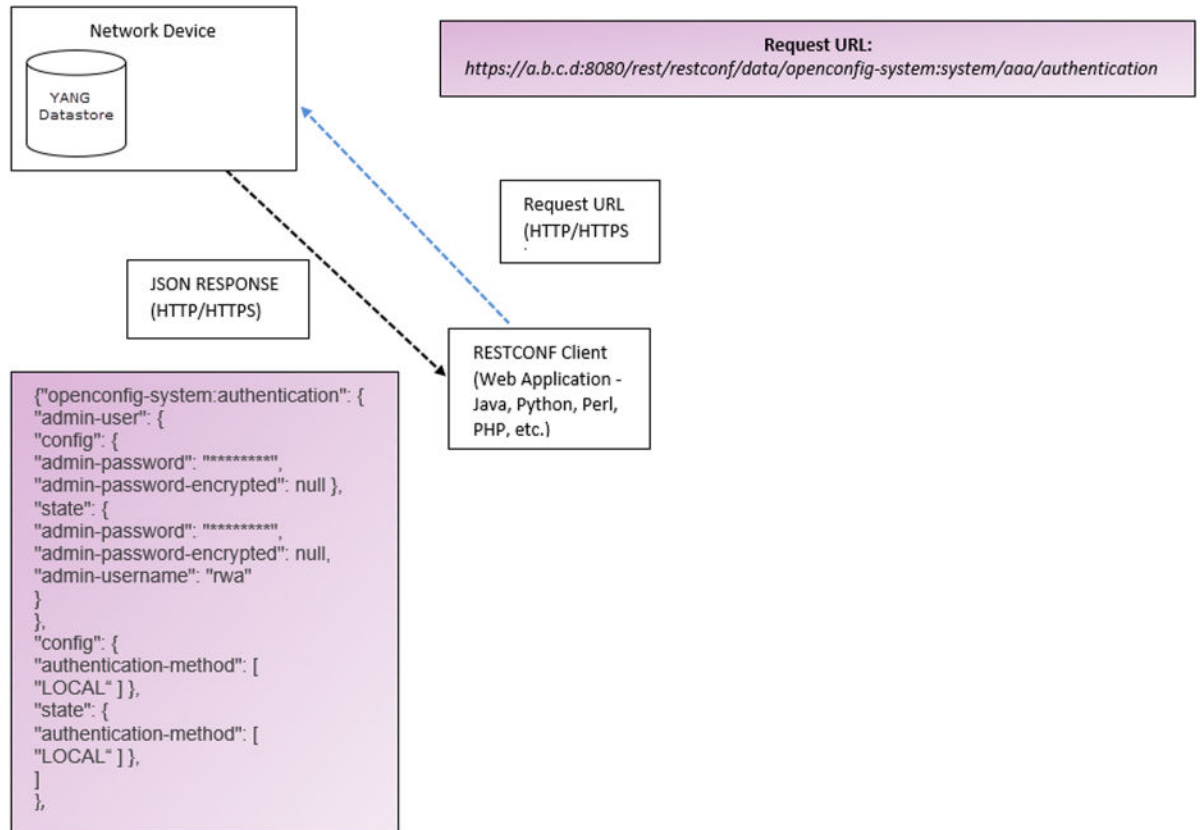
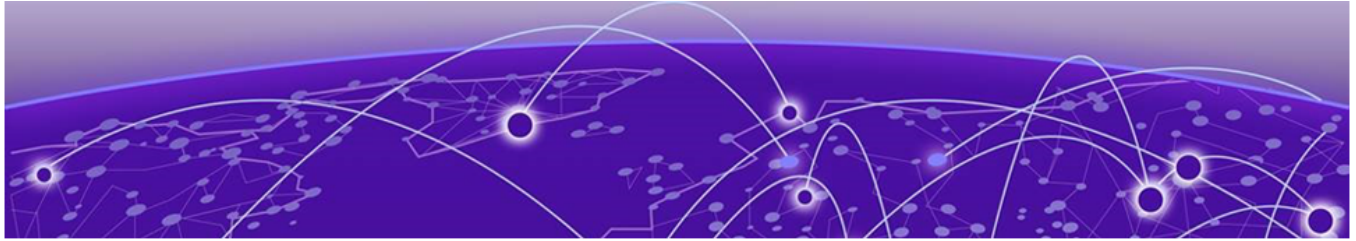


Figure 1: RESTCONF Protocol Architecture



YANG Data Models

[JSON Representation](#) on page 16

YANG is a data modeling language used to model configuration and state data as published in [RFC 6020](#). The Representational State Transfer Configuration Protocol (RESTCONF) interface supports YANG models defined by standards bodies and community groups such as [IETF](#) and [OpenConfig](#), as well as native YANG models. Some aspects of the data model to note:

- Every data model is a **module**, a self-contained top-level hierarchy of nodes.
- The data model uses **containers** to group related nodes.
- The data model uses **lists** to identify nodes that are stored in sequence.
- Each individual attribute of a node is represented by a **leaf**.
- Every leaf must have an associated **data type**.

```
module: openconfig-aaa
  +--rwaaa
  | +--rwconfig
  | +--rostate
  | +--rwauthentication
  | | +--rwconfig
  | | | +--rwauthentication-method* union
  | | | +--rostate
  | | | +--roauthentication-method* union
  | | +--rwadmin-user
  | | | +--rwconfig
  | | | | +--rwadmin-password? string
  | | | | +--rwadmin-password-hashed? oc-aaa-types:crypt-password-type
  | | | +--rostate
  | | | +--roadmin-password? string
  | | | +--roadmin-password-hashed? oc-aaa-types:crypt-password-type
  | | | +--roadmin-username? String
```

Figure 2: YANG Data Model Components

Callout	Description
1	Module name
2	Data type
3	Leaf
4	List
5	Container

To see the published YANG model tree, URLs for each supported YANG model, and the complete JSON for the YANG model, go to [RESTCONF Reference Documentation](#).

Modules	
OpenConfig	Relay Agent (DHCP support)
OpenConfig	Interfaces Port: POE, port attributes, such as auto-sense, default-vlan-id, flex-uni, qos, untag-port-default-vlan
OpenConfig	Interfaces LAG: attributes, such as flex-uni
OpenConfig	Interfaces VLAN: IPv4 configuration
OpenConfig	Platform: ports, CPU, fans, power supply, optical devices - GET operations only
OpenConfig	Network Instance: VLAN interface - VRF association, CVLAN I-SID, IS-IS redistribute direct, IPVPN, I-SID, and IP DHCP relay forward path)
OpenConfig	STP: STP global information and port interface bpduguard state, RSTP global and port level information, MSTP global, MST instance level state - GET operations only
OpenConfig	System (aaa)
OpenConfig	LLDP
OpenConfig	VLAN

Table 4: IETF YANG Data Models

Data model	Description
ietf-interfaces	Contains a collection of YANG definitions for managing network interfaces.

Table 5: Extreme Enterprise YANG Data Models

Data model	Description
extreme-virtual-service	Defines data for managing Extreme virtual services.
extreme-network-service	Defines data for managing Extreme network services.

JSON Representation

VOSS and Fabric Engine support JSON format to represent the data resource. The following information describes the JSON representation for the YANG elements:

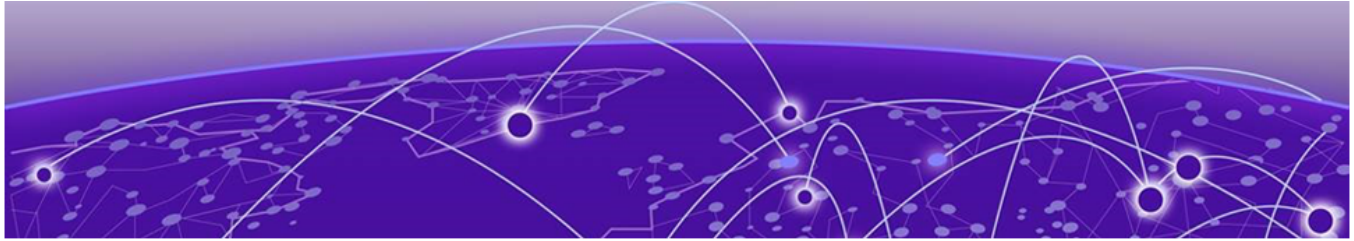
- The YANG elements in the resource models are mapped to JSON elements for the proper serialization.
- A leaf element is mapped into a single key-value pair. The key and the value are separated by a colon.

- A container element is mapped into a JSON object. Thus, the equivalent representation of a container starts with a left curly bracket and ends with a right curly bracket. The elements within the container are separated a comma.
- A list element is mapped into a JSON array. Thus, the equivalent representation of the list starts with a left square bracket and ends with a right square bracket. The instances of the list element are separated by a comma.

The following is an example of JSON representation.

```
{
  "openconfig-system:authentication":{
    "admin-user":{
      "config":{
        "admin-password": "*****",
        "admin-password-encrypted": null },
      "state": {
        "admin-password": "*****",
        "admin-password-encrypted": null },
        "admin-username": "rwa"
      }
    },
    "config":{
      "authentication-method": [
        "LOCAL" ]
    },
    "state": {
      "authentication-method": [
        "LOCAL" ]
    },
    "users": { "user": [{"config": {
      "password": "*****",
      "password-encrypted": null,
      "role": "read-write-all",
      "ssh-key": null,
      "username": "user1"},
      "state": {"password": "*****",
        "password-encrypted": null,
        "role": "read-write-all",
        "ssh-key": null,
        "username": "user1"},
      "username": "user1"},{
      "config": {"password": "*****",
        "password-encrypted": null,
        "role": "read-only",
        "ssh-key": null,
        "username": "ro"},
      "state": {"password": "*****",
        "password-encrypted": null,
        "role": "read-only",
        "ssh-key": null,
        "username": "ro"},
      "username": "ro"}, {
      "config": {"password": "*****",
        "password-encrypted": null,
        "role": "read-write",
        "ssh-key": null,
        "username": "rw"},
      "state": {
        "password": "*****",
        "password-encrypted": null,
        "role": "read-write",
        "ssh-key": null,
```

```
        "username": "rw"},
    "username": "rw"}, {
    "config": {
        "password": "*****",
        "password-encrypted": null,
        "role": "SYSTEM_ROLE_ADMIN",
        "ssh-key": null,
        "username": "rwa"},
    "state": {
        "password": "*****",
        "password-encrypted": null,
        "role": "SYSTEM_ROLE_ADMIN",
        "ssh-key": null,
        "username": "rwa"},
    "username": "rwa"}}}
    }
}
```

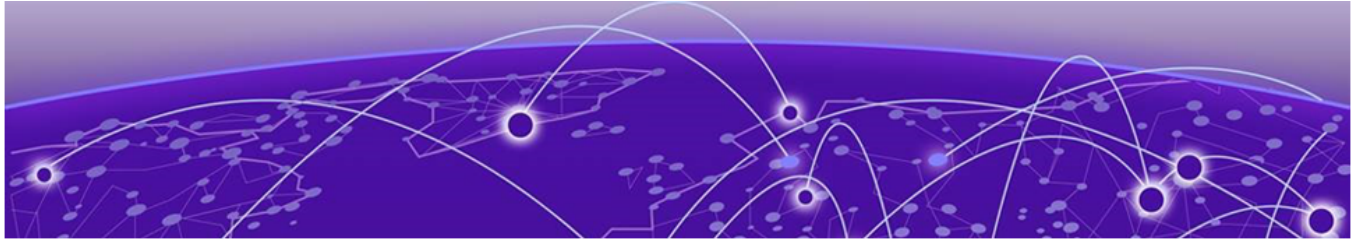


CRUD Operations

The Representational State Transfer Configuration Protocol (RESTCONF) API supports the **GET**, **POST**, **PATCH**, and **DELETE** HTTP methods.

Table 6: RESTCONF API CRUD Methods

Method	Description	Usage
GET	Sent by the client to retrieve data and metadata for a resource.	If the GET method succeeds, response message body will have the JSON result.
POST	Sent by the client to create a data resource or invoke an operation resource.	If the POST method succeeds, a "201 Created" status-line is returned and there is no response message body. If the data resource already exists, then the POST request must fail and "409 Conflict" status-line is returned.
PATCH	Can be used by the client to create or update a target resource. Merges the contents of the message body with the target resource.	If the PATCH request succeeds, a "200 OK" status-line is returned if there is a message body, and "204 No Content" is returned if no message body is sent.
DELETE	Used by the client to delete the target resource.	If the DELETE request succeeds, a "204 No Content" status-line is returned.



Query Parameters

[depth](#) on page 21
[content](#) on page 28
[insert](#) on page 33
[point](#) on page 36
[formatted](#) on page 38
[filter](#) on page 39

Each Representational State Transfer Configuration Protocol (RESTCONF) operation allows zero or more [query parameters](#) to be present in the request URL. The specific parameters allowed depends on the resource type, and sometimes the specific target resource used, in the request.

- Query parameters can be given in any order.
- Each parameter can appear only one time in a request URL.
- A default value can apply if the parameter is missing.
- Query parameter names and values are case-sensitive.
- A server **MUST** return an error with a `400 Bad Request` status-line if a query parameter is unexpected.
- The value of any query parameter **MUST** be encoded according to RFC3986. Any reserved characters **MUST** be percent-encoded, according to [RFC 3986](#).

Table 7: Supported RESTCONF Query Parameters

Name	Methods	Description
content on page 28	GET, HEAD	Select config or non-config data resources.
depth on page 21	GET, HEAD	Request limited subtree depth in the reply content.
filter on page 39	GET, HEAD	Return a human-readable JSON formatted response instead of a non-formatted block of data.
formatted on page 38	GET, HEAD	Reduces the response data by only returning the filtered result.
insert on page 33	GET, HEAD	Insertion mode for <i>ordered-by-user</i> data resources.
point on page 36	GET, HEAD	Insertion point for <i>ordered-by-user data</i> resources.

depth

The depth query parameter

The **depth** query parameter is used to limit the number of nested levels returned by the server. Data nodes with a **depth** value greater than the **depth** parameter are not returned in a response for a **GET** method.

- The value of the **depth** parameter will be either an integer between 1 and 65535, or the string unbounded. The default is unbounded, which retrieves all child resources.
- The first nest-level will be the requested data node.
- The **depth** parameter is allowed only for **GET** methods on API datastore and data resources. A 400 Bad Request status is returned if used for other methods or resource types.

Sample request from client to retrieve the `interfaces` YANG model data without specifying the **depth** parameter

```
GET /rest/restconf/data/openconfig-interfaces:interfaces
HTTP/1.1
Host: 10.68.5.64
```

Sample response data from server

```
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "config": {
          "description": "X440G2-24t-G4 Port 1",
          "enabled": true,
          "mtu": 1500,
          "name": "1",
          "type": "ethernetCsmacd"
        },
        "hold-time": {
          "config": {
            "down": 0,
            "up": 0
          },
          "state": {
            "down": 0,
            "up": 0
          }
        },
        "name": "1",
        "openconfig-if-ethernet:ethernet": {
          "config": {
            "auto-negotiate": true,
            "duplex-mode": "FULL",
            "enable-flow-control": false,
            "mac-address": "00:04:96:9E:4B:80",
            "port-speed": "SPEED_UNKNOWN"
          },
          "openconfig-vlan:switched-vlan": {
            "config": {
```

```

        "access-vlan": 1,
        "interface-mode": "ACCESS"
    },
    "state": {
        "access-vlan": 1,
        "interface-mode": "ACCESS"
    }
},
"state": {
    "auto-negotiate": true,
    "counters": {
        "in-8021q-frames": 0,
        "in-crc-errors": 0,
        "in-fragment-frames": 0,
        "in-jabber-frames": 0,
        "in-mac-control-frames": 0,
        "in-mac-pause-frames": 0,
        "in-oversize-frames": 0,
        "out-8021q-frames": 0,
        "out-mac-control-frames": 0,
        "out-mac-pause-frames": 0
    },
    "duplex-mode": "FULL",
    "effective-speed": 0,
    "enable-flow-control": false,
    "hw-mac-address": "00:04:96:9E:4B:80",
    "mac-address": "00:04:96:9E:4B:80",
    "negotiated-duplex-mode": "FULL",
    "negotiated-port-speed": "SPEED_UNKNOWN",
    "port-speed": "SPEED_UNKNOWN"
}
},
"state": {
    "admin-status": "UP",
    "counters": {
        "in-broadcast-pkts": 0,
        "in-discards": 0,
        "in-errors": 0,
        "in-multicast-pkts": 0,
        "in-octets": 0,
        "in-unicast-pkts": 0,
        "in-unknown-protos": 0,
        "last-clear": "2017-12-19T14:25:53Z",
        "out-broadcast-pkts": 0,
        "out-discards": 0,
        "out-errors": 0,
        "out-multicast-pkts": 0,
        "out-octets": 0,
        "out-unicast-pkts": 0
    },
    "description": "X440G2-24t-G4 Port 1",
    "enabled": true,
    "ifindex": 1001,
    "last-change": 0,
    "mtu": 1500,
    "name": "1",
    "openconfig-platform-transceiver:physical-channel": [],
    "openconfig-platform:hardware-port": "00:04:96:9E:4B:80",
    "oper-status": "DOWN",
    "type": "ethernetCsmacd"
},
"subinterfaces": {
    "subinterface": [
        {

```

```

    "config": {
      "description": "",
      "enabled": true,
      "index": 1000004,
      "name": "Default"
    },
    "index": 1000004,
    "openconfig-vlan:vlan": {
      "config": {
        "vlan-id": 1
      },
      "state": {
        "vlan-id": 1
      }
    },
    "state": {
      "admin-status": "UP",
      "counters": {
        "in-broadcast-pkts": 0,
        "in-discards": 0,
        "in-errors": 0,
        "in-multicast-pkts": 0,
        "in-octets": 0,
        "in-unicast-pkts": 0,
        "in-unknown-protos": 0,
        "out-broadcast-pkts": 0,
        "out-discards": 0,
        "out-errors": 0,
        "out-multicast-pkts": 0,
        "out-octets": 0,
        "out-unicast-pkts": 0
      },
      "description": "",
      "enabled": true,
      "ifindex": 1000004,
      "index": 1000004,
      "last-change": 4400,
      "name": "Default",
      "oper-status": "DOWN"
    }
  }
]
},
},

```

This can be more detail than the management application needs. Specifying the **depth** parameter trims the response.

Sample request from client with **depth** = 1

```

GET rest/restconf/data/openconfig-interfaces:interfaces?depth=1
HTTP/1.1
Host: 10.68.5.64

```

Sample response data from server

```

{
  "openconfig-interfaces:interfaces": {}
}

```

Sample request from client with **depth** = 2

```
GET /rest/restconf/data/openconfig-interfaces:interfaces?depth=2
HTTP/1.1
Host: 10.68.5.64
```

Sample response data from server

```
{
  "openconfig-interfaces:interfaces": {
    "interface": {}
  }
}
```

Sample request from client with **depth** = 3

```
GET /rest/restconf/data/openconfig-interfaces:interfaces?depth=3
HTTP/1.1
Host: 10.68.5.64
```

Using **depth** = 3 gives you a top level listing of all the interfaces and their types. The query runs faster on the device because it does not have to collect all the details of the full interfaces YANG model.

Sample response from server

```
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "config": {},
        "hold-time": {},
        "name": "1",
        "openconfig-if-ethernet:ethernet": {},
        "state": {},
        "subinterfaces": {}
      },
      {
        "config": {},
        "hold-time": {},
        "name": "2",
        "openconfig-if-ethernet:ethernet": {},
        "state": {},
        "subinterfaces": {}
      },
      {
        "config": {},
        "hold-time": {},
        "name": "3",
        "openconfig-if-ethernet:ethernet": {},
        "state": {},
        "subinterfaces": {}
      },
      {
        "config": {},
        "hold-time": {},
        "name": "4",
```

```
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "5",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "6",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "7",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "8",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "9",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "10",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "11",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "12",
```

```
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "13",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "14",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "15",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "16",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "17",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "18",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "19",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "20",
```

```
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "21",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "22",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "23",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "24",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "25",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "26",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "27",
    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "hold-time": {},
    "name": "28",
```

```

    "openconfig-if-ethernet:ethernet": {},
    "state": {},
    "subinterfaces": {}
  },
  {
    "config": {},
    "name": "Default",
    "state": {},
    "subinterfaces": {}
  }
]
}
}

```

content

The content query parameter

The **content** query parameter controls how descendant nodes of the requested data nodes will be processed in the reply. The allowed values are:

Table 8: Content Query Parameter Values

Value	Description
config	Return only configuration descendant data nodes
nonconfig	Return only non-configuration descendant data nodes
all	Return all descendant data nodes

The **content** parameter is allowed only for **GET** methods on datastore and data resources. A 400 Bad Request status is returned if used for other methods or resource types.

If the **content** query parameter is not present in the request, the default value is **all**.

Sample request from client with **content** = config

```

GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1?content=config
HTTP/1.1
Host: 10.68.5.64

```

Sample response data from the server

```

{
  "openconfig-interfaces:interface": [
    {
      "config": {
        "description": "",
        "enabled": true,
        "mtu": 1500,
        "name": "1",
        "type": "ethernetCsmacd"
      },
      "hold-time": {
        "config": {

```

```

        "down": 0,
        "up": 0
      }
    },
    "name": "1",
    "openconfig-if-ethernet:ethernet": {
      "config": {
        "auto-negotiate": true,
        "duplex-mode": "FULL",
        "enable-flow-control": false,
        "mac-address": "00:04:96:9A:4F:4F",
        "port-speed": "SPEED_UNKNOWN"
      },
      "openconfig-vlan:switched-vlan": {
        "config": {
          "access-vlan": 1,
          "interface-mode": "ACCESS"
        }
      }
    },
    "subinterfaces": {
      "subinterface": [
        {
          "config": {
            "description": "",
            "enabled": true,
            "index": 1000004,
            "name": "Default"
          },
          "index": 1000004
        }
      ]
    }
  }
}
]]

```

Sample request from client with **content** = nonconfig

```

GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1?content=nonconfig
HTTP/1.1
Host: 10.68.5.64

```

Sample response data from server

```

{
  "openconfig-interfaces:interface": [
    {
      "hold-time": {
        "state": {
          "down": 0,
          "up": 0
        }
      },
      "name": "1",
      "openconfig-if-ethernet:ethernet": {
        "openconfig-vlan:switched-vlan": {
          "state": {
            "access-vlan": 1,
            "interface-mode": "ACCESS"
          }
        }
      }
    }
  ]
}

```

```

"state": {
  "auto-negotiate": true,
  "counters": {
    "in-8021q-frames": 0,
    "in-crc-errors": 0,
    "in-fragment-frames": 0,
    "in-jabber-frames": 0,
    "in-mac-control-frames": 0,
    "in-mac-pause-frames": 0,
    "in-oversize-frames": 0,
    "out-8021q-frames": 0,
    "out-mac-control-frames": 0,
    "out-mac-pause-frames": 0
  },
  "duplex-mode": "FULL",
  "effective-speed": 0,
  "enable-flow-control": false,
  "hw-mac-address": "00:04:96:9A:4F:4F",
  "mac-address": "00:04:96:9A:4F:4F",
  "negotiated-duplex-mode": null,
  "negotiated-port-speed": "SPEED_UNKNOWN",
  "port-speed": "SPEED_UNKNOWN"
},
"state": {
  "admin-status": "UP",
  "counters": {
    "in-broadcast-pkts": 0,
    "in-discards": 0,
    "in-errors": 0,
    "in-multicast-pkts": 0,
    "in-octets": 0,
    "in-unicast-pkts": 0,
    "in-unknown-protos": 0,
    "last-clear": "2020-03-16T07:40:46Z",
    "out-broadcast-pkts": 0,
    "out-discards": 0,
    "out-errors": 0,
    "out-multicast-pkts": 0,
    "out-octets": 0,
    "out-unicast-pkts": 0
  },
  "description": "",
  "enabled": true,
  "ifindex": 1001,
  "last-change": 3800,
  "mtu": 1500,
  "name": "1",
  "openconfig-platform-transceiver:physical-channel": [],
  "openconfig-platform:hardware-port": "00:04:96:9A:4F:4F",
  "oper-status": "DOWN",
  "type": "ethernetCsmacd"
},
"subinterfaces": {
  "subinterface": [
    {
      "index": 1000004,
      "state": {
        "admin-status": "UP",
        "counters": {
          "in-broadcast-pkts": 0,
          "in-discards": 0,
          "in-errors": 0,
          "in-multicast-pkts": 0,

```

```

        "in-octets": 0,
        "in-unicast-pkts": 0,
        "in-unknown-protos": 0,
        "out-broadcast-pkts": 0,
        "out-discards": 0,
        "out-errors": 0,
        "out-multicast-pkts": 0,
        "out-octets": 0,
        "out-unicast-pkts": 0
      },
      "description": "",
      "enabled": true,
      "ifindex": 1000004,
      "index": 1000004,
      "last-change": 3800,
      "name": "Default",
      "oper-status": "DOWN"
    }
  ]
}
1}

```

Sample request from client with **content** = all

```

GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1?content=all
HTTP/1.1
Host: 10.68.5.64

```

...which is the same as

```

GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1
HTTP/1.1
Host: 10.68.5.64

```

Sample response data from server

```

{
  "openconfig-interfaces:interface": [
    {
      "config": {
        "description": "",
        "enabled": true,
        "mtu": 1500,
        "name": "1",
        "type": "ethernetCsmacd"
      },
      "hold-time": {
        "config": {
          "down": 0,
          "up": 0
        },
        "state": {
          "down": 0,
          "up": 0
        }
      },
      "name": "1",
    }
  ]
}

```

```

"openconfig-if-ethernet:ethernet": {
  "config": {
    "auto-negotiate": true,
    "duplex-mode": "FULL",
    "enable-flow-control": false,
    "mac-address": "00:04:96:9A:4F:4F",
    "port-speed": "SPEED_UNKNOWN"
  },
  "openconfig-vlan:switched-vlan": {
    "config": {
      "access-vlan": 1,
      "interface-mode": "ACCESS"
    },
    "state": {
      "access-vlan": 1,
      "interface-mode": "ACCESS"
    }
  },
  "state": {
    "auto-negotiate": true,
    "counters": {
      "in-8021q-frames": 0,
      "in-crc-errors": 0,
      "in-fragment-frames": 0,
      "in-jabber-frames": 0,
      "in-mac-control-frames": 0,
      "in-mac-pause-frames": 0,
      "in-oversize-frames": 0,
      "out-8021q-frames": 0,
      "out-mac-control-frames": 0,
      "out-mac-pause-frames": 0
    },
    "duplex-mode": "FULL",
    "effective-speed": 0,
    "enable-flow-control": false,
    "hw-mac-address": "00:04:96:9A:4F:4F",
    "mac-address": "00:04:96:9A:4F:4F",
    "negotiated-duplex-mode": null,
    "negotiated-port-speed": "SPEED_UNKNOWN",
    "port-speed": "SPEED_UNKNOWN"
  },
  "state": {
    "admin-status": "UP",
    "counters": {
      "in-broadcast-pkts": 0,
      "in-discards": 0,
      "in-errors": 0,
      "in-multicast-pkts": 0,
      "in-octets": 0,
      "in-unicast-pkts": 0,
      "in-unknown-protos": 0,
      "last-clear": "2020-03-16T07:40:46Z",
      "out-broadcast-pkts": 0,
      "out-discards": 0,
      "out-errors": 0,
      "out-multicast-pkts": 0,
      "out-octets": 0,
      "out-unicast-pkts": 0
    },
    "description": "",
    "enabled": true,
    "ifindex": 1001,
    "last-change": 3800,
  }
}

```

```

    "mtu": 1500,
    "name": "1",
    "openconfig-platform-transceiver:physical-channel": [],
    "openconfig-platform:hardware-port": "00:04:96:9A:4F:4F",
    "oper-status": "DOWN",
    "type": "ethernetCsmacd"
  },
  "subinterfaces": {
    "subinterface": [
      {
        "config": {
          "description": "",
          "enabled": true,
          "index": 1000004,
          "name": "Default"
        },
        "index": 1000004,
        "state": {
          "admin-status": "UP",
          "counters": {
            "in-broadcast-pkts": 0,
            "in-discards": 0,
            "in-errors": 0,
            "in-multicast-pkts": 0,
            "in-octets": 0,
            "in-unicast-pkts": 0,
            "in-unknown-protos": 0,
            "out-broadcast-pkts": 0,
            "out-discards": 0,
            "out-errors": 0,
            "out-multicast-pkts": 0,
            "out-octets": 0,
            "out-unicast-pkts": 0
          },
          "description": "",
          "enabled": true,
          "ifindex": 1000004,
          "index": 1000004,
          "last-change": 3800,
          "name": "Default",
          "oper-status": "DOWN"
        }
      }
    ]
  }
}

```

insert

The insert query parameter

You can use the **insert** query parameter to specify how a resource should be inserted within an *ordered-by user* list. The possible values are:

Table 9: Insert Query Parameter Values

Value	Description
first	Insert the new data as the new first entry
last	Insert the new data as the new last entry

Table 9: Insert Query Parameter Values (continued)

Value	Description
before	Insert the new data before the insertion point, as specified by the value of the point parameter
after	Insert the new data after the insertion point, as specified by the value of the point parameter

The default value is **last**.



Note

The **insert** parameter is supported only for the **POST** and **PUT** methods. It is also only supported if the target resource is a data resource, and that data represents a YANG list or leaf-list that is *ordered-by user*.

A **point** query parameter MUST also be present for the **insert** query parameter if the values **before** or **after** are used, otherwise a 400 Bad Request status is returned.

Sample request from client to insert an access control entry at the head of the list

```
POST /rest/restconf/data/ietf-access-control-list:acls/acl=acl/aces?insert=first
HTTP/1.1
Host: 10.50.130.172
Content-Type: application/yang-data+json
{
  ietf-access-control-list:aces": [
    {
      "matches": {
        "eth": {
          "ethertype": 0
        }
      },
      "name": "ace0",
      "actions": {
        "forwarding": "drop"
      }
    }
  ]
}
```

Sample response data from server

```
{
  "ietf-access-control-list:aces": {
    "ace": [
      {
        "actions": {
          "forwarding": "drop"
        },
        "matches": {
          "eth": {
            "ethertype": 0
          }
        }
      },

```

```

    "name": "ace0",
    "statistics": {
      "matched-packets": 0
    }
  },
  {
    "actions": {
      "forwarding": "drop"
    },
    "matches": {
      "eth": {
        "ethertype": 1
      }
    },
    "name": "ace1",
    "statistics": {
      "matched-packets": 0
    }
  },
  {
    "actions": {
      "forwarding": "drop"
    },
    "matches": {
      "eth": {
        "ethertype": 288
      }
    },
    "name": "ace2",
    "statistics": {
      "matched-packets": 0
    }
  },
  {
    "actions": {
      "forwarding": "drop"
    },
    "matches": {
      "eth": {
        "ethertype": 3
      }
    },
    "name": "ace3",
    "statistics": {
      "matched-packets": 0
    }
  },
  {
    "actions": {
      "forwarding": "drop"
    },
    "matches": {
      "eth": {
        "ethertype": 4
      }
    },
    "name": "ace4",
    "statistics": {
      "matched-packets": 0
    }
  }
]
}

```

point

The point query parameter

You can use the **point** query parameter in conjunction with the **insert** parameter to specify the insertion point for a data resource that is being created or moved within an *ordered-by user* list or leaf-list.

The value of the **point** parameter is a string that identifies the path to the insertion point object. The format is the same as a target resource URI string.



Note

The **point** parameter is supported only for the **POST** and **PUT** methods. It is also only supported if the target resource is a data resource, and that data represents a YANG list or leaf-list that is *ordered-by user*.

If the **insert** query parameter is not present or has a value other than **before** or **after**, then a 400 Bad Request status is returned.

Sample request from client to insert a new access control entry in the access control list after ace1

```
POST /rest/restconf/data/ietf-access-control-list:acls/acl=acl/aces?
insert=after&point=ace1
HTTP/1.1
Host: 10.50.130.172
Content-Type: application/yang-data+json

{
  "ietf-access-control-list:aces": [
    {
      "actions": {
        "forwarding": "drop"
      },
      "matches": {
        "eth": {
          "ethertype": 2
        }
      },
      "name": "ace2"
    }
  ]
}
```

Sample response data from server

```
{
  "ietf-access-control-list:aces": {
    "ace": [
      {
        "actions": {
          "forwarding": "drop"
        },
        "matches": {
          "eth": {
            "ethertype": 65535
          }
        }
      }
    ]
  }
}
```

```

    }
  },
  "name": "aceOVERFLOW",
  "statistics": {
    "matched-packets": 0
  }
},
{
  "actions": {
    "forwarding": "drop"
  },
  "matches": {
    "eth": {
      "ethertype": 0
    }
  },
  "name": "ace0",
  "statistics": {
    "matched-packets": 0
  }
},
{
  "actions": {
    "forwarding": "drop"
  },
  "matches": {
    "eth": {
      "ethertype": 1
    }
  },
  "name": "ace1",
  "statistics": {
    "matched-packets": 0
  }
},
{
  "actions": {
    "forwarding": "drop"
  },
  "matches": {
    "eth": {
      "ethertype": 288
    }
  },
  "name": "ace2",
  "statistics": {
    "matched-packets": 0
  }
},
{
  "actions": {
    "forwarding": "drop"
  },
  "matches": {
    "eth": {
      "ethertype": 3
    }
  },
  "name": "ace3",
  "statistics": {
    "matched-packets": 0
  }
},
{

```

```
    "actions": {
      "forwarding": "drop"
    },
    "matches": {
      "eth": {
        "ethertype": 4
      }
    },
    "name": "ace4",
    "statistics": {
      "matched-packets": 0
    }
  }
]
}
```

formatted

The formatted query parameter

You can use the **formatted** query parameter to return a human-readable JSON formatted response instead of a non-formatted block of data.

If the formatted query parameter is not present in the request, the default value is FALSE.

Sample request from client with the default behavior of FALSE

```
GET /rest/restconf/data/openconfiginterfaces:interfaces/interface=1/config
HTTP/1.1
Host: 10.68.5.80
```

Sample response data from server

```
{"openconfig-interfaces:config":
{"enabled":true,"type":"ethernetCsmacd","description":"","name":"1","mtu":1500}}
```

Sample request

```
GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1/config?formatted=True
HTTP/1.1
Host: 10.68.5.80
```

Sample response data from server

```
{
  "openconfig-interfaces:config": {
    "enabled": true,
    "type": "ethernetCsmacd",
    "description": "",
    "name": "1",
    "mtu": 1500
  }
}
```

```
}  
}
```

filter

The filter query parameter

The **filter** query parameter is used to specify a response with certain data fields.

Using the **filter** reduces the response data by only returning the filtered results.

You must use JSONPath with the **filter** query parameter. More information on JSONPath can be found at <https://goessner.net/articles/JsonPath/> and <https://jsonpath.com/>.

Sample request from client to filter for all interface names - ports and VLANs

```
GET /rest/restconf/data/openconfig-interfaces:interfaces?  
formatted=True&filter=$.'openconfig-interfaces:interfaces'.interface[*].name  
HTTP/1.1  
Host: 10.68.5.80
```

Sample response data from server

```
[  
  "1",  
  "10",  
  "11",  
  "12",  
  "13",  
  "14",  
  "15",  
  "16",  
  "17",  
  "18",  
  "19",  
  "2",  
  "20",  
  "21",  
  "22",  
  "23",  
  "24",  
  "25",  
  "26",  
  "27",  
  "28",  
  "29",  
  "3",  
  "30",  
  "31",  
  "32",  
  "33",  
  "34",  
  "4",  
  "5",  
  "6",  
  "7",  
  "8",  
]
```

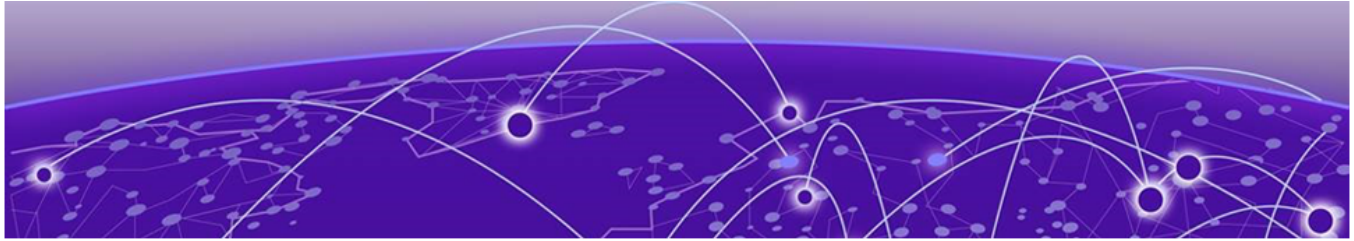
```
"9",  
"Default",  
"TestVlan1234",  
"interdut",  
"testbfdlbpk",  
"testbfdlbpk2",  
"testbfdvlan",  
]
```

Sample request to filter for only port interface names

```
GET /rest/restconf/data/openconfig-interfaces:interfaces?formatted=True&$.'openconfig-  
interfaces:interfaces'.interface[?(@.state.type=='ethernetCsmacd')].name  
HTTP/1.1  
Host: 10.68.5.80
```

Sample response data from server

```
[  
"1",  
"10",  
"11",  
"12",  
"13",  
"14",  
"15",  
"16",  
"17",  
"18",  
"19",  
"2",  
"20",  
"21",  
"22",  
"23",  
"24",  
"25",  
"26",  
"27",  
"28",  
"29",  
"3",  
"30",  
"31",  
"32",  
"33",  
"34",  
"4",  
"5",  
"6",  
"7",  
"8",  
"9"  
]
```

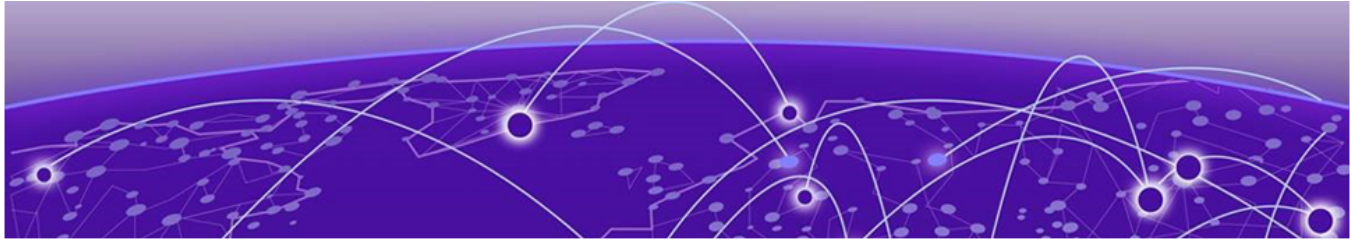


Error Response Codes

The Representational State Transfer Configuration Protocol (RESTCONF) API returns standard HTTP status codes in addition to JSON-based error codes and messages in the response body.

Table 10: HTTP status codes

Code	Description
200 OK	The request was successful.
201 Created	The resource was created successfully.
204 No Content	Success with no response body.
400 Bad Request	The operation failed because the request is syntactically incorrect or violated schema.
401 Unauthorized	The authentication credentials are invalid or the user is not authorized to use the API.
404 Not Found	The server did not find the specified resource that matches the request URL.
405 Method Not Allowed	The API does not support the requested HTTP method.
409 Data Resource Already Exists	The data resource already exists.



Log in to the RESTCONF Server

[RESTCONF Root Resource](#) on page 42

[Authentication](#) on page 43

[Access the RERESTCONFSTCONF Datastores](#) on page 43

You must have admin access to the network device that is running the Representational State Transfer Configuration Protocol (RESTCONF) interface. No separate login credentials are required.

RESTCONF listens on the IP address assigned to the device. When unencrypted (that is, without SSL), it operates on the default HTTP port 80. For encrypted connections, it operates on the HTTPS port 443.

RESTCONF Root Resource

[RFC 8040](#) requires that Representational State Transfer Configuration Protocol (RESTCONF) interfaces have a common URL as the root URL. When first connecting to a RESTCONF server, a RESTCONF client must determine the root of the RESTCONF API. To support data integrity and confidentiality, RESTCONF requires HTTPS. The root resource for RESTCONF is:

```
/rest/restconf/
```

This is determined by sending the following **GET** request to the RESTCONF server:

```
GET /.well-known/host-meta HTTP/1.1
Host: <Device_IP_Address>
Accept: application/xrd+xml
```

The server responds as follows:

```
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn

<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
  <Link rel='restconf' href='/rest/restconf' />
</XRD>
```



Note

The datastore is represented by a node named `data`. All methods are supported on `data`.

Authentication

About This Task

You must start a valid session by sending a basic authentication request to the Representational State Transfer Configuration Protocol (RESTCONF) server, before you can start making API calls. There are no limits on the number of requests per session.

Procedure

1. Use the **POST** method to send the initial login request to the RESTCONF API server.

Sample client request

```
curl -k -d '{"username":"rwa","password":"rwa"}'
-H "Content-Type: application/json"
-X POST http://<Device_IP_Address>:8080/auth/token
```

2. The RESTCONF server responds as follows:

Sample server response

```
{"token": "IjcyNzc2MS03Mjc3NjEtNmM2ZjYzNjE2Yy0zMTMwMmUzMjJlMzMzOTJlMzUzOC1.
YtUL8Q.cBUfBcF5FumXQTE1Dp0Ak0VHnY"}
```



Note

The token is valid for a day and can be used to send subsequent requests. Include the token in the request header for all subsequent API calls.

Including the authorization token as a cookie in the request header

```
curl -X GET http://<Device_IP_Address>/rest/restconf/data/openconfig_vlan:vlan=1/
config \
-H 'Content-Type: application/json' \
-H 'Cookie: x-auth-
token=eyJhbGciOiJIUzI1NiIsImV4cCI6MTU3MDE5MDk3OSwiaWF0IjoxNTcwMTA0NTc5fQ.
eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNTcwMTA0NTc5fQ.
I2uaCxQ8v5-ShAaZHwUbs76e9LZ22Who4icgLBwmVc8'
```

Access the RERESTCONFSTCONF Datastores

About This Task

Unlike REST implementations, Representational State Transfer Configuration Protocol (RESTCONF) offers deterministic URI strings and JSON formatting based on YANG data models. To retrieve data or configure a device using the RESTCONF interface, you must use the proper URI string to access the resource in question. Each YANG module defines a hierarchy of data that you can use to retrieve the URI and the exact parameters accepted by the JSON payload for RESTCONF-based operations.

A RESTCONF URI is encoded from left to right, starting from the root to the target resource:

```
{+restconf}/data/<yang-module:container>/<leaf>[?<query_parameters>]
```

- `{+restconf}/data` is the root resource for the combined configuration and state data resources that can be accessed by a client, where `{+restconf}` indicates the root URL for the device.
- `<yang-module:container>` is the base model container being used.
- `<leaf>` is an individual element from within the container.
- Some network devices can support options sent as `<query_parameters>` that impact returned results.

For example, to access the top-level `interfaces` resource within the `openconfig-interfaces` YANG model, the URL would be:

```
https://<ip>/rest/restconf/data/openconfig-interfaces:interfaces.
```

To access the interfaces data model and collect the data on interface=1 (port 1) on the device:

Procedure

1. Use the **GET** method to retrieve details of the specific interface:

Sample client request

```
GET /rest/restconf/data/openconfig-interfaces:interfaces/interface=1
HTTP/1.1
Host: 10.68.13.192
```

2. The server responds as follows:

Sample server response

```
{
  "openconfig-interfaces:interface": [
    {
      "config": {
        "description": "Port1",
        "enabled": false,
        "mtu": 1500,
        "name": "1",
        "type": "ethernetCsmacd"
      },
      "hold-time": {
        "config": {
          "down": 0,
          "up": 0
        },
        "state": {
          "down": 0,
          "up": 0
        }
      },
      "name": "1",
      "openconfig-if-ethernet:ethernet": {
        "config": {
          "auto-negotiate": true,
          "duplex-mode": "FULL",
          "enable-flow-control": false,
          "mac-address": "00:11:88:FE:AE:98",
```

```

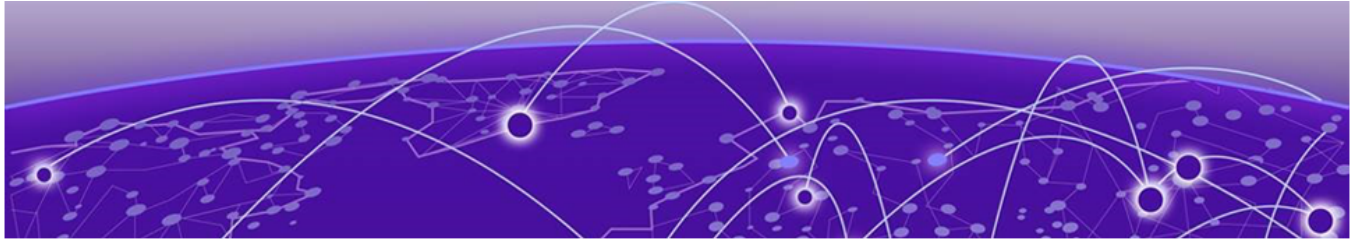
    "port-speed": "SPEED_1GB"
  },
  "openconfig-if-poe:poe": {
    "config": {
      "enabled": true
    },
    "state": {
      "enabled": true,
      "power-class": 0,
      "power-used": 0.0
    }
  },
  "openconfig-vlan:switched-vlan": {
    "config": {
      "access-vlan": 1,
      "interface-mode": "ACCESS"
    },
    "state": {
      "access-vlan": 1,
      "interface-mode": "ACCESS"
    }
  },
  "state": {
    "auto-negotiate": true,
    "counters": {
      "in-8021q-frames": 0,
      "in-crc-errors": 0,
      "in-fragment-frames": 0,
      "in-jabber-frames": 0,
      "in-mac-control-frames": 0,
      "in-mac-pause-frames": 0,
      "in-oversize-frames": 0,
      "out-8021q-frames": 0,
      "out-mac-control-frames": 0,
      "out-mac-pause-frames": 0
    },
    "duplex-mode": "FULL",
    "effective-speed": 0,
    "enable-flow-control": false,
    "hw-mac-address": "00:11:88:FE:AE:98",
    "mac-address": "00:11:88:FE:AE:98",
    "negotiated-port-speed": "SPEED_UNKNOWN",
    "port-speed": "SPEED_1GB"
  }
},
"state": {
  "admin-status": "DOWN",
  "counters": {
    "in-broadcast-pkts": 0,
    "in-discards": 0,
    "in-errors": 0,
    "in-multicast-pkts": 0,
    "in-octets": 0,
    "in-unicast-pkts": 0,
    "in-unknown-protos": 0,
    "last-clear": "2020-02-10T22:06:43Z",
    "out-broadcast-pkts": 0,
    "out-discards": 0,
    "out-errors": 0,
    "out-multicast-pkts": 0,
    "out-octets": 0,
    "out-unicast-pkts": 0
  },
  "description": "Port1",

```

```

        "enabled": false,
        "ifindex": 1001,
        "last-change": 6500,
        "mtu": 1500,
        "name": "1",
        "openconfig-platform-transceiver:physical-channel": [],
        "openconfig-platform:hardware-port": "00:11:88:FE:AE:98",
        "oper-status": "DOWN",
        "type": "ethernetCsmacd"
    },
    "subinterfaces": {
        "subinterface": [
            {
                "config": {
                    "description": "",
                    "enabled": true,
                    "index": 1000004,
                    "name": "Default"
                },
                "index": 1000004,
                "state": {
                    "admin-status": "UP",
                    "counters": {
                        "in-broadcast-pkts": 0,
                        "in-discards": 0,
                        "in-errors": 0,
                        "in-multicast-pkts": 0,
                        "in-octets": 0,
                        "in-unicast-pkts": 0,
                        "in-unknown-protos": 0,
                        "out-broadcast-pkts": 0,
                        "out-discards": 0,
                        "out-errors": 0,
                        "out-multicast-pkts": 0,
                        "out-octets": 0,
                        "out-unicast-pkts": 0
                    },
                    "description": "",
                    "enabled": true,
                    "ifindex": 1000004,
                    "index": 1000004,
                    "last-change": 7900,
                    "name": "Default",
                    "oper-status": "UP"
                }
            }
        ]
    }
}

```



RESTCONF API and CLI Commands Examples

- [Port CLI Commands](#) on page 47
- [RESTCONF API for Port CLI Commands](#) on page 48
- [MLT CLI Commands](#) on page 49
- [RESTCONF API for MLT CLI Command](#) on page 49
- [VLAN CLI Commands](#) on page 50
- [RESTCONF API for VLAN CLI Command](#) on page 50
- [VRF CLI Commands](#) on page 51
- [RESTCONF API for VRF CLI Commands](#) on page 52
- [I-SID CLI Commands](#) on page 58
- [RESTCONF API for I-SID CLI Command](#) on page 59

The following section provides the Representational State Transfer Configuration Protocol (RESTCONF) API section for each CLI command.

Port CLI Commands

This section provides the list of applicable port CLI commands.

- ```
interface gigabitEthernet <port-list>
 [no] encapsulation dot1q
exit
```
- ```
interface gigabitEthernet <port-list>
default-vlan-id <vid>
exit
```
- ```
interface gigabitEthernet <port-list>
[no] untag-port-default-vlan
exit
```
- ```
interface gigabitEthernet <port-list>
flex-uni enable
exit
```
- ```
interface gigabitEthernet <port-list>
[no] shutdown
exit
```

- ```
interface gigabitEthernet <port-list>
  qos if-shaper shape-rate 234000
  exit
```
- ```
interface gigabitEthernet <port-list>
 auto-sense enable
 exit
```

## RESTCONF API for Port CLI Commands

This section provides the Representational State Transfer Configuration Protocol (RESTCONF) API sections for the port CLI commands.

### PATCH

- To access the top-level interfaces resource within the openconfig-interfaces YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/
config
```

Use this API operation to do the following:

- Enable or disable Flex-UNI.
- Configure QoS shape rate.
- Configure Auto-sense.

Sample server response:

```
{
 "openconfig-interfaces:config":
 {
 "flex-uni": true,
 "qos-shape-rate": 10,
 "auto-sense": true
 }
}
```

- To access the top-level interfaces resource within the openconfig-interfaces YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/
ethernet/switched-vlan/config
```

Use this API operation to do the following:

- Configure a default VLAN ID.
- Enable or disable the default VLAN on a tagged port.

Sample server response:

```
{
 "openconfig-interfaces:config":
 {
 "native-vlan": 1, -> "default-vlan-id <vid>" command on port
 interface
 }
}
```

```

 "untag-port-default-vlan": true,
 }
}

```

## GET

To access the top-level interfaces resource within the openconfig-interfaces YANG model, use the following URLs:

```
URL: https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/
config
```

```
URL: https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/
state
```

Use this API operation to do the following:

- Get interface configuration.
- Get URLs available in RESTCONF Phase 1.

Sample server response:

Based on Yang Model

## MLT CLI Commands

This section provides the list of applicable MLT CLI commands.

- ```

interface mlt <mlt-id>
flex-uni enable
exit

```

RESTCONF API for MLT CLI Command

This section provides the Representational State Transfer Configuration Protocol (RESTCONF) API sections for the MLT CLI command.

POST or PATCH

- To access the top-level interfaces resource within the openconfig-interfaces YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/
config
```

Use this API operation to do the following:

- Enable or disable Flex-UNI.

Sample server response:

```

{
  "openconfig-interfaces:config":
  {

```

```

        "flex-uni": true,
    }
}

```

GET

To access the top-level interfaces resource within the openconfig-interfaces YANG model, use the following URLs:

```

https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/config
https://$ip:8080/rest/restconf/data/openconfig-interfaces:interfaces/interface=%s/state

```

Use this API operation to do the following:

- Get interface configuration.
- Get URLs available in RESTCONF Phase 1.

Sample server response:

Based on Yang Model

VLAN CLI Commands

This section provides the list of applicable VLAN CLI commands.

- ```

interface vlan 1101
 vrf mgmt
 ip address 10.128.147.X 255.255.255.252
 ip dhcp-relay
 ip dhcp-relay circuitId
 ip dhcp-relay remoteId
exit

```

## RESTCONF API for VLAN CLI Command

This section provides the Representational State Transfer Configuration Protocol (RESTCONF) API sections for the VLAN CLI command.

## POST or PATCH

- To access the top-level interfaces resource within the openconfig-relay-agent YANG model, use the following URL:

```

https://$ip:8080/rest/restconf/data/openconfig-relay-agent: relay-agent/dhcp/
interfaces/interface=<vlan-id>/agent-information-option

```

Use this API operation to do the following:

- Configure DHCP Relay information on a VLAN.

Sample server response:

```

{
 " openconfig-relay-agent: agent-information-option":

```

```

 {
 "config":
 {
 "enable": "<true | false>",
 "circuit-id": "<circuit id>",
 "remote-id": "<remote id>"
 }
 }
 }
{

```

## GET

To access the top-level interfaces resource within the openconfig-relay-agent YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-relay-agent: relay-agent/dhcp/interfaces/
interface=<vlan-id>/agent-information-option
```

Use this API operation to do the following:

- Get DHCP Relay configuration on a VLAN interface.

Sample server response:

Based on Yang Model

## DELETE

To access the top-level interfaces resource within the openconfig-relay-agent YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-relay-agent: relay-agent/dhcp/interfaces/
interface=<vlan-id>
```

Use this API operation to do the following:

- Remove DHCP Relay configuration on a VLAN interface.

## VRF CLI Commands

This section provides the list of applicable VRF CLI commands.

- ```

ip vrf <name>
router vrf cip
  ipvpn
  i-sid 2000
  ipvpn enable
  isis redistribute direct
  isis redistribute direct metric 1
  isis redistribute direct enable
  ip dhcp-relay fwd-path 10.128.147.X 10.128.0.48
  ip dhcp-relay fwd-path 10.128.147.X 10.128.0.48 enable
  ip dhcp-relay fwd-path 10.128.147.X 10.128.0.48 mode dhcp
exit

```
- ```

isis apply redistribute direct vrf cip

```

- ```

interface vlan 1101
  vrf cmgmt
  exit
vlan i-sid <vid> <i-sid>

```

RESTCONF API for VRF CLI Commands

This section provides the Representational State Transfer Configuration Protocol (RESTCONF) API sections for the VRF CLI commands.

POST

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```

https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/config

```

Use this API operation to do the following:

- Create a VRF and configure parameters, such as VRF name, IP VPN status, and I-SID.
- Configure an IP VPN to enabled does both (CLI): **ipvpn** and **ipvpn enable**.

Sample server response:

```

{
  "openconfig-network-instance:config":
  {
    "name": "<VRF name>",
    "ipvpn": "<IpVpn status>",
    "nsi": "<network service ID(I-SID)>"
  }
}

```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```

https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s

```

Use this API operation to do the following:

- Create a VRF and configure parameters, such as VRF name, IP VPN status, I-SID.
- Configure an IP VPN to enabled does both (CLI): **ipvpn** and **ipvpn enable**.

Sample server response:

```

{
  "openconfig-network-instance:network-instance":
  {
    "config":
    {
      "name": "<VRF name>",
      "ipvpn": "<IpVpn status>",
      "nsi": "<network service ID(I-SID)>"
    }
  }
}

```

```

    }
  }
}

```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/vlans/vlan=%s/config
```

Use this API operation to do the following:

- Associate an existing VLAN with VRF.
- Create an I-SID of type CVLAN. In CLI: **vlan i-sid <vid> <i-sid>**

Sample server response:

```

{
  "openconfig-network-instance:config ":
  {
    "nsi":<c-vlan i-sid>
  }
}

```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/vlans/vlan=<VLAN ID>
```

Use this API operation to do the following:

- Associate an existing VLAN with VRF.
- Create an I-SID of type CVLAN. In CLI: **vlan i-sid <vid> <i-sid>**

Sample server response:

```

{
  "openconfig-network-instance:vlan ":
  {
    "config":
    {
      "nsi":<c-vlan i-sid>
    }
  }
}

```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/vlans
```

Use this API operation to do the following:

- Associate an existing VLAN with VRF.
- Create an I-SID of type CVLAN. In CLI: **vlan i-sid <vid> <i-sid>**

Sample server response:

```
{
  "openconfig-network-instance:vlan ":
  {
    "vlan":
    {
      "config":
      {
        "vlan-id": <VLAN ID>,
        "nsi":<c-vlan i-sid>
      }
    }
  }
}
```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=<VRF ID or name>/
dhcp-relay/forward-path=<agent address>%26<server address>/config
```

Use this API operation to do the following:

- Configure DHCP-relay in a VRF instance.
- Create a new forward path.

Forward path is passed in the URL as: <agent address>%26server address>.

%26 is & URL encoded.

For example:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=1/dhcp-relay/forward-path=10.1.1.1%2610.1.1
```

Sample server response:

```
{
  "openconfig-network-instance:config":
  {
    "agent-address ": "<agent address>",
    "server-address ": "<server address>",
    "enable": "<true | false>",
    "mode": "DHCP | BOOTP | BOTH"
  }
}
```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/dhcp-relay/forward-path=<agent address>%26<server address>
```

Use this API operation to do the following:

- Configure DHCP-relay in a VRF instance.
- Create a new forward path.

Forward path is passed in the URL as: <agent address>%26server address>.

%26 is & URL encoded.

Sample server response:

```
{
  "openconfig-network-instance:config":
  {
    "agent-address ": "<agent address>",
    "server-address ": "<server address>",
    "enable": "<true | false>",
    "mode": "DHCP | BOOTP | BOTH"
  }
}
```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/dhcp-relay
```

- Use this API operation to do the following:
 - Configure DHCP-relay in a VRF instance.
 - Create a new forward path.

Sample server response:

```
{
  "openconfig-network-instance:config":
  {
    "forward-path ":
    {
      "config":
      {
        "agent-address ": "<agent address>",
        "server-address ": "<server address>",
        "enable": "<true | false>",
        "mode": "DHCP | BOOTP | BOTH"
      }
    }
  }
}
```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/protocols/protocol=%s/config
```

- Use this API operation to do the following:
 - Configure IS-IS redistribute parameters.

Protocol instance is: identifier (protocol type: DIRECTLY_CONNECTED) + name (a unique name for the protocol instance).

enabled parameter does (CLI): **isis redistribute direct enable** on router VRF + **isis apply redistribute direct vrf cip**.

Sample server response:

```
{
  "openconfig-network-instance:config":
  {
    "enabled": "<true | false>",
    "default-metric": "<metric>"
  }
}
```

GET

To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URLs:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance=%s
```

- >VRF name/VFR ID

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance=%s/config
```

- >VRF name/VFR ID

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance=%s/state
```

- >VRF name/VFR ID

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

= VRF name or name

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/vlans

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/vlans/vlan=<VLAN ID>

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/vlans/vlan=<VLAN ID>/state

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/vlans/vlan=<VLAN ID>/config

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/protocols/protocol=<protocol identifier>

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/protocols/protocol=<protocol identifier>/state

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/protocols/protocol=<protocol identifier>/config

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/dhcp-relay



Note

DHCP relay forward path identifier: agent + server addresses, is passed through URL as: server address>%26<agent address>. Where %26 is encoded value for &.

For example:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance=<VRF ID or name>/dhcp-relay/forward-path=<agent address>%26<server address>
```

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/dhcp-relay/forward-path=<server address>%26<agent address>

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/dhcp-relay/forward-path=<server address>%26<agent address>/config

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance
```

=<VRF ID or name>/dhcp-relay/forward-path=<server address>%26<agent address>/state

Use this API operation to do the following:

- Get VRF configuration.

Sample server response:

Based on Yang Model

DELETE

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/network-instance=%s
```

Use this API operation to do the following:

- Remove VRF.
- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/vlans/vlan=%d
```

Use this API operation to do the following:

- Remove VLAN association with a given VRF.
- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/dhcp-relay/forward-path=%s
```

Use this API operation to do the following:

- Remove VLAN association with a given VRF.
- DHCP relay forward path identifier: agent + server addresses, is passed through URL as: server address>%26<agent address>. Where %26 is encoded value for &.

For example:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=<VRF ID or name>/dhcp-relay/forward-path=<agent address>%26<server
address>
```

- To access the top-level interfaces resource within the openconfig-network-instance YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/openconfig-network-instance:network-instances/
network-instance=%s/protocols/protocol=%s
```

Use this API operation to do the following:

- Remove IS-IS redistribute configuration on a given VRF.

I-SID CLI Commands

This section provides the list of applicable I-SID CLI commands.

- CVLAN

```
vlan i-sid <vid> <i-sid>
i-sid name <i-sid> <name>
no vlan i-sid <vid>
```

- Switched UNI (ELAN) and Transparent UNI

```
i-sid <i-sid> elan
[no] c-vid <vid> port <port-list>
exit
```

```
i-sid <i-sid> elan
```

```
[no] c-vid <vid> mlt <mlt-id>
exit
```

```
i-sid <i-sid> elan
[no] untagged-traffic port <port-list> [bpdu enable]
exit
```

```
i-sid <i-sid> elan
[no] untagged-traffic mlt <mlt-id> [bpdu enable]
exit
```

```
i-sid <i-sid> elan- transparent
[no] port <port-list>
exit
```

```
i-sid <i-sid> elan- transparent
[no] mlt <mlt-id>
exit
```

RESTCONF API for I-SID CLI Command

This section provides the Representational State Transfer Configuration Protocol (RESTCONF) API sections for the I-SID CLI command.

POST

- To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-service=<I-SID>
```

Use this API operation to do the following:

- Create an I-SID with a given name and type (CVLAN = I2vsn, Switched UNI (S-UNI) = elan, Transparent Port UNI (T-UNI) = elan_transparent)
- Associate a CVLAN, an S-UNI, and a T-UNI with an interface (VLAN, port, or LAG).
- Add CVID and BPDU configuration on interfaces associated with an S-UNI I-SID.

Sample server response:

```
{
  "extreme-network-service:network-service":
  [
    {
      "id": "< service id>",
      "type": "<i-sid type: elan, elan_transparent, I2vsn>",
      "name": "<i-sid name>",
      "interfaces":
      {
        "interface":
        {
```

```

        "name": "<interface name>",
        "endpoints":
        {
            "endpoint" :
            {
                "interface-name": "<interface name>",
                "cvid": "<cvid>",
                "bpdu": "<bpdu>"
            }
        }
    }
}

```

Create a CVLAN by associating the VLAN to an I-SID:

```

{
  "extreme-network-service:network-service":
  [
    {
      "id": "2",
      "interfaces":
      {
        "interface":
        [
          {
            "name": "VLAN-2"
          }
        ]
      },
      "name": "isid2",
      "type": "l2vsn"
    }
  ]
}

```

Create a T-UNI by associating the MLT to an I-SID:

```

{
  "extreme-network-service:network-service":
  [
    {
      "id": "3",
      "interfaces":
      {
        "interface":
        [
          {
            "name": "MLT-3"
          }
        ]
      },
      "name": "isid3",
      "type": "elantransparent"
    }
  ]
}

```

Create an S-UNI I-SID with a VLAN:

```
{
  "extreme-network-service:network-service":
  [
    {
      "id": "6",
      "interfaces":
      {
        "interface":
        [
          {
            "name": "VLAN-3"
          }
        ]
      },
      "name": "isid6",
      "type": "elan"
    }
  ]
}
```

Add a CVID and an MLT to the S-UNI:

```
{
  "extreme-network-service:network-service":
  [
    {
      "id": "6",
      "interfaces":
      {
        "interface":
        [
          {
            "endpoints"
            {
              "endpoint" :
              {
                "bpdu": "true",
                "cvid": "4096",
                "interface-name": "MLT-7",
              }
            },
            "name": "MLT-7",
          }
        ],
        "name": "isid6666",
        "type": "elan"
      }
    }
  ]
}
```

PATCH

- To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URL:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-service=<I-SID>
```

Use this API operation to do the following:

- Reconfigure an I-SID name.
- Associate an S-UNI and a T-UNI with another interface (VLAN, port, or LAG).
- Add CVID and BPDU configuration on interfaces associated with an S-UNI I-SID.



Note

The new CVID is added to the existing configuration.

Sample server response:

```
{
  "extreme-network-service:network-service":
  [
    {
      "name": "<i-sid name>",
      "interfaces":
      [
        {
          "interface":
          {
            "name": "<interface name>",
            "endpoints":
            [
              {
                "endpoint" :
                {
                  "interface-name": "<interface name>",
                  "cvid": "<cvid>",
                  "bpdu": "<bpdu>"
                }
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Associate a T-UNI I-SID with a port interface:

```
{
  "extreme-network-service:network-service":
  [
    {
      "id": "3",
      "interfaces":
      [
        {
          "interface":
          [
            {
              "name": "1/5"
            }
          ]
        },
        {
          "name": "ISID-3",
          "type": "elanTransparent"
        }
      ]
    }
  ]
}
```

Associate an S-UNI I-SID with a port interface and a CVID:

```
{
  "extreme-network-service:network-service":
  [
    {
      "id": "6",
      "interfaces":
      {
        "interface":
        {
          "endpoints":
          {
            "endpoint":
            {
              "bpdu": "true",
              "cvid": "1",
              "interface-name": "1/6"
            }
          }
        },
        "name": "1/6"
      },
      "name": "isid6666",
      "type": "elan"
    }
  ]
}
```

GET

To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URLs:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s/interfaces
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s/interfaces/interface=%s
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s/interfaces/interface=%s/endpoints
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s/interfaces/interface=%s/endpoints/endpoint=%s
```

DELETE

- To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URLs:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-
service=%s
```

Use this API operation to do the following:

- Remove an I-SID.
- To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URLs:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-service=%s/interfaces/interface=%s
```

Use this API operation to do the following:

- Remove interface (VLAN, port, or LAG) association with an I-SID.
- To access the top-level interfaces resource within the openconfig-extreme-network-services YANG model, use the following URLs:

```
https://$ip:8080/rest/restconf/data/extreme-network-service:network-services/network-service=%s/interfaces/interface=%s/endpoints/endpoint=%s
```

- Use this API operation to do the following:
 - Remove endpoint for S-UNI and T-UNI.