ExtremeXOS® and Switch Engine™ RESTCONF API Developer Guide

for Version 32.1

9037493-00 Rev AA
June 2022

Copyright © 2022 Extreme Networks, Inc. All rights reserved.

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: www.extremenetworks.com/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/

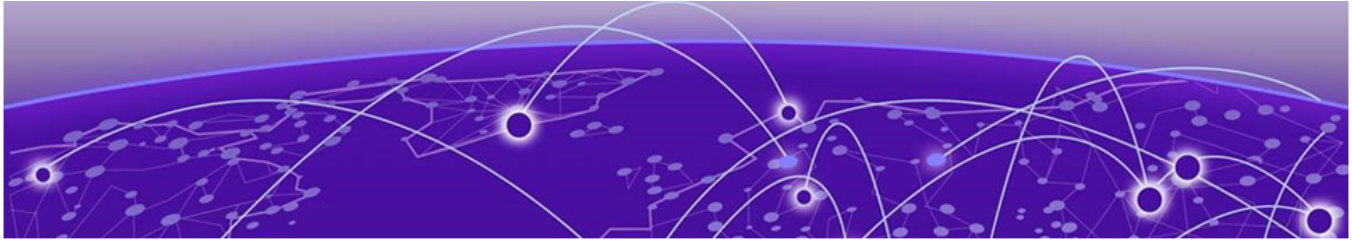


Table of Contents

Preface **4**
- Text Conventions 4
- Documentation and Training 5
- Help and Support 6
  - Subscribe to Product Announcements 6
- Send Feedback 6

RESTCONF Interface **8**
- RESTCONF Protocol 8
- YANG Data Models 9
  - JSON Representation 14
- CRUD Operations 15
- Output Compression 16
- Query Parameters 16
  - depth 17
  - content 24
  - insert 30
  - point 32
  - formatted 34
  - filter 34
- Error Response Codes 36

Log in to the RESTCONF Server **38**
- RESTCONF Root Resource 38
- Authentication 39
- Access the RESTCONF Datastores 39

API Usage Examples **43**
- Get VLAN Details 43
- Create a VLAN 44
- Change VLAN Settings 44
- Delete a VLAN 45

Preface

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.

Text Conventions

Unless otherwise noted, information in this document applies to all supported environments for the products in question. Exceptions, like command keywords associated with a specific software version, are identified in the text.

When a feature, function, or operation pertains to a specific hardware product, the product name is used. When features, functions, and operations are the same across an entire product family, such as ExtremeSwitching switches or SLX routers, the product is referred to as *the switch* or *the router*.

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

Convention	Description
`screen displays`	This typeface indicates command syntax, or represents information as it is displayed on the screen.
The words *enter* and *type*	When you see the word *enter* 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 *type*.
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**
Words in italicized type	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.
NEW!	New information. In a PDF, this is searchable text.

Table 3: Command syntax

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

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 white papers, data sheets, and case studies

Extreme Networks offers product training courses, both online and in person, as well as specialized certifications. For details, visit www.extremenetworks.com/education/.

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 2826. 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 Information Development 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, do either of the following:

- Access the feedback form at https://www.extremenetworks.com/documentation-feedback/.
- Email us at documentation@extremenetworks.com.

Provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.

RESTCONF Interface

RESTCONF Protocol on page 8
YANG Data Models on page 9
CRUD Operations on page 15
Output Compression on page 16
Query Parameters on page 16
Error Response Codes on page 36

This document is released with ExtremeXOS and Switch Engine 32.1.

Note
RESTCONF can be enabled for ExtremeXOS 22.1 and later. It is bundled with ExtremeXOS beginning with 22.4, and Switch Engine beginning with 31.6.

Note
For ExtremeXOS versions between 22.1 and 22.4, you can download the RESTCONF files here: https://github.com/extremenetworks/EXOS_Apps/tree/master/REST/downloads

For complete release information, refer to *ExtremeXOS Release Notes*.

RESTCONF Protocol

Representational State Transfer Configuration (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 heterogenous 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.

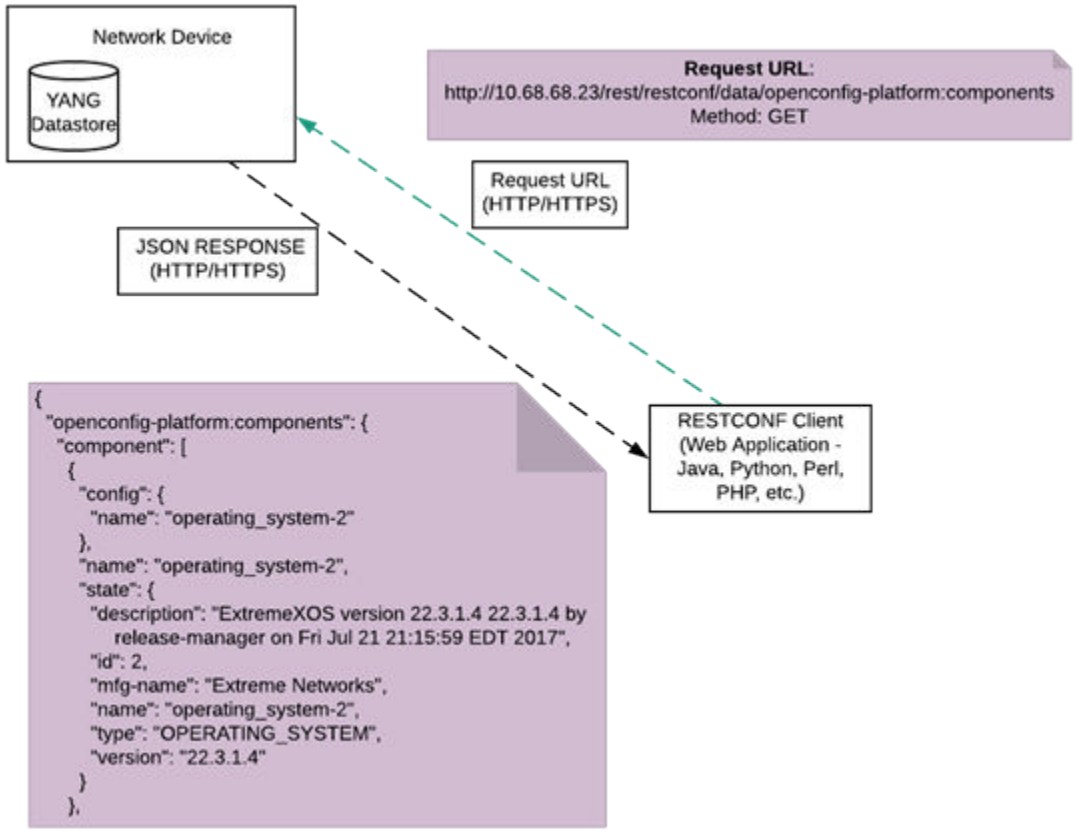


Figure 1: RESTCONF Protocol Architecture

Related Topics

ExtremeXOS RESTCONF API Reference
RESTCONF - RFC 8040
YANG - RFC 6020
YANG Data Types - RFC 6021

YANG Data Models

YANG is a data modeling language used to model configuration and state data as published in RFC 6020. The 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**.

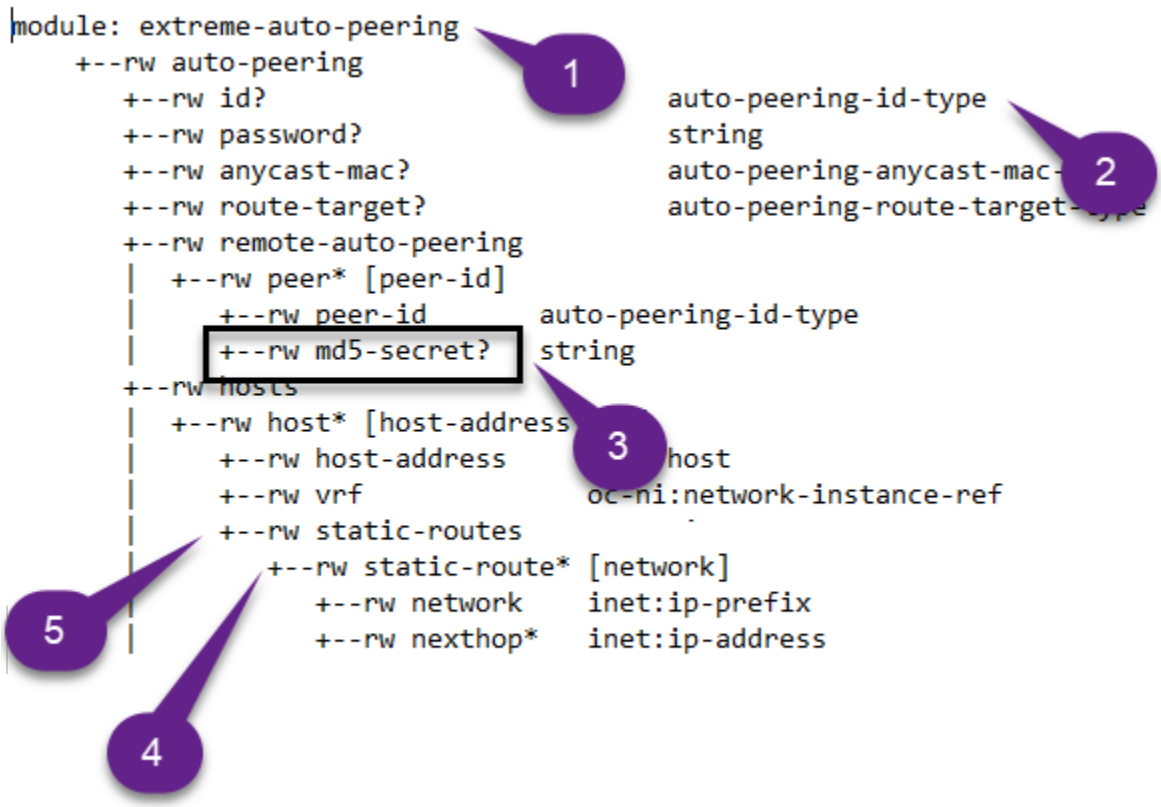


Figure 2: YANG Data Model Components

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

Callout	Description
5	Container

Note

To see the published YANG model tree, URLs for each supported YANG model, and the complete JSON for the YANG model, go to RESTCONF API Reference.

Table 4: OpenConfig YANG Data Models

Data model	Description
openconfig-bgp	Defines a limited subset of the BGP protocol configuration data.
openconfig-interfaces	Defines data for managing network interfaces and subinterfaces. The following modules are supported: • `IPv[46]`: This module defines data for managing configuration and operation state on IP (IPv4 and IPv6) interfaces. • `eth`: This module defines data for managing Ethernet interfaces. • `lag`: This module defines data for managing aggregated interfaces (bundle, LAG).
openconfig-lacp	Defines configuration and operational state data for the Link Aggregation Control Protocol (LACP).
openconfig-lldp	Defines configuration and operational state data for the Link Layer Discovery Protocol (LLDP).
openconfig-module-catalog	Provides a schema for cataloging and describing YANG models published across various organizations. The catalog contains several categories of data: • `organizations`: Entities that publish or maintain individual YANG modules or groups of modules. • `modules`: Information about individual YANG modules, including their versions, dependencies, submodules, and how to access them. • `release bundles`: Groups of modules that are compatible and consistent with each other. The release bundle does not necessarily correspond to a functional area; it could contain the entire set of modules published by an organization. • `feature bundles`: Sets of schema paths across a release bundle that provide a specific set of functionality. • `implementations`: Information about available module or bundle implementations and their status.

Table 4: OpenConfig YANG Data Models (continued)

Data model	Description
openconfig-network-instance	Provides an OpenConfig description of a network instance. This can be a Layer 3 forwarding construct such as virtual routing and forwarding (VRF) instance, or a Layer 2 instance such as a virtual switch instance (VSI). The following modules are supported: • `vlans`: This module defines configuration and state variables for VLANs, in addition to VLAN parameters associated with interfaces. • `rib-bgp`: This module defines data for representing BGP routing table or Routing Information Base (RIB) contents. The model supports five logical RIBs per address family. • `ospf`: This module defines data for representing and managing Open Shortest Path First (OSPF) versions 2 and 3.
openconfig-platform	Defines a data model for representing a system component inventory, which can include hardware or software elements arranged in an arbitrary structure. The primary relationship supported by the model is containment (for example, components containing subcomponents). The properties for each component can include dynamic values. For example, a CPU component may report its utilization, temperature, or other physical properties.
openconfig-relay-agent	Describes configuration and operational state data related to relay agents typically used for DHCP and BOOTP packets. It supports both DHCP and DHCPv6, and device-wide and per-interface settings.
openconfig-telemetry	Creates the configuration for the telemetry systems and functions on the device, to monitor data continuously with efficient, incremental updates.
openconfig-spanning-tree	Defines configuration and operational state data for the Spanning Tree Protocol.

Table 4: OpenConfig YANG Data Models (continued)

Data model	Description
openconfig-system	Defines data for managing system-wide services and functions on network devices. The following modules are supported: • `aaa`: This module defines configuration and operational state data related to authorization, authentication, and accounting (AAA) management. • `telemetry`: This module creates the configuration for the telemetry systems and functions on the device, to monitor data continuously with efficient, incremental updates.
openconfig-vlan	Defines configuration and state variables for VLANs, in addition to VLAN parameters associated with interfaces.

Table 5: IETF YANG Data Models

Data model	Description
ietf-access-control-list	Defines configuration and operational state data for the Access Control Lists (ACLs).
ietf-interfaces	Contains a collection of YANG definitions for managing network interfaces.
ietf-yang-library	Contains monitoring information about the YANG modules and submodules that are used within a YANG-based server.

Table 6: Extreme Enterprise YANG Data Models

Data model	Description
extreme-auto-peering	Defines the auto peering mechanism for cooperating interconnected devices to form a network of any scale for any topology, providing fully redundant, multipath routing.
extreme_edge_automation	Defines model for managing Extreme Edge Automation.
extreme-fabric	Provides the top-level model for the configuration and operational state of Extreme fabrics.
extreme-virtual-service	Defines data for managing Extreme virtual services.
extreme-acl-usage	Defines Access Control List rules and usage.
extreme-policy-cos	Describes Policy Class of Service feature.

Table 6: Extreme Enterprise YANG Data Models (continued)

Data model	Description
extreme-dns-cache	Describes the configuration and operational state data for DNS Cache and DNS Cache Analytics feature provided by Extreme Networks. DNS Cache is a mechanism of caching DNS queries and responses routed through the switch. By default, DNS queries are resolved by forwarding the query to the configured DNS servers and a response is received. Based on the response, the DNS cache is built. For further queries, the DNS cache is referred to resolve instead of forwarding to the DNS server. DNS Cache Analytics analyzes the DNS query logs (both IPv4 and IPv6). Analyzed data maintains a list of DNS queries from connected clients, domain names accessed by clients, and the time stamps of each DNS query.
extreme-license	Defines data model for the unified license feature provided by Extreme Networks.

JSON Representation

ExtremeXOS and Switch 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.

```
{
    "extreme-fabric:fabric":{
        "spb-fabric":{
            "ipv6-src-address":"",
            "enable":false,
            "ip-src-address":"",
            "system-id":"",
            "manual-area":"",
            "instances":{
                "instance":[
                    {
                        "smlt":{
                            "smlt-peer":"",
                            "smlt-peer-system-id":""
                        },
                        "ip-shortcuts":{
                            "ipv4-enable":false,
                            "ipv6-enable":false
                        },
```

```
                        "number":99,
                        "lsdb-trap":false,
                        "multicast":{
                            "enable":false,
                            "fwd-cache-timeout":0
                        },
                        "redistribute-direct":{
                            "global-router":false,
                            "vrf-all":false
                        },
                        "equal-cost-trees":{
                            "equal-cost-tree":[
                                {
                                    "ect":"ONE",
                                    "id":1
                                }
                            ]
                        },
                        "nickname":"",
                        "spb-vlans":{
                            "spb-vlan":[
                                {
                                    "id":99
                                }
                            ]
                        }
                    }
                ]
            },
            "cfm":{
                "set-admin":false,
                "mep-id":0,"level":0
            },
...
```

CRUD Operations

The RESTCONF API supports the **GET**, **POST**, **PATCH**, and **DELETE** HTTP methods.

Table 7: 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 resouce.	If the DELETE request succeeds, a "204 No Content" status-line is returned.

NEW! Output Compression

The RESTCONF API supports the gzip-output compression scheme. When interfacing into RESTCONF through scripts, you can use `Accept-Encoding` to specify the gzip compression data.

Sample request from client

```
GET /encrypted-area HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
```

Sample response data from server

```
HTTP/1.1 200 OK
Date: mon, 26 June 2016 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
```

Sample request from client using 'curl'

```
curl -X GET
http://10.68.5.80/rest/restconf/data/openconfig_vlan:vlans
\
-H 'Content-Type: application/json' \
-H 'Cookie: x-authtoken=
eyJhbGciOiJIUzI1NiIsImV4cCI6MTYyODU1NjUwMSwiaWF0IjoxNjI4NDcwM
TAxfQ.eyJ1c2VybmFtZSI6ImFkbWluIiwiYWNjZXNzX2xldmVsIjoiYWRtaW4ifQ.x1
nEONl-jIbcrjNuTAUnS0y-a-_gtMsHdMQEMur09JY' \
-H 'Accept-Encoding: gzip, deflate' \
| gunzip -
```

Query Parameters

Each RESTCONF operation allows zero or more query parameters to be present in the request URL. The specific parameters allowed will depend 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 once in a request URI.
- A default value may 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 8: Supported RESTCONF Query Parameters

Name	Methods	Description
content on page 24	GET, HEAD	Select config or non-config data resources
depth on page 17	GET, HEAD	Request limited subtree depth in the reply content
filter on page 34	GET, HEAD	Return a human-readable JSON formatted response instead of a non-formatted block of data
formatted on page 34	GET, HEAD	Reduces the response data by only returning the filtered results
insert on page 30	GET, HEAD	Insertion mode for *ordered-by-user* data resources
point on page 32	GET, HEAD	Insertion point for *ordered-by-user data* resources

depth

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"
              }
            }
          ]
        }
```

```
      },
and so on...
```

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

Table 9: 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"
            }
           }
          ]
        }
    }
]}
```

*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

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 10: 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
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

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
        }
      }
    ]
  }
}
```

NEW! formatted

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":"","n
ame":"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
    }
}
```

NEW! filter

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",
"testbfdlpbk",
"testbfdlpbk2",
"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

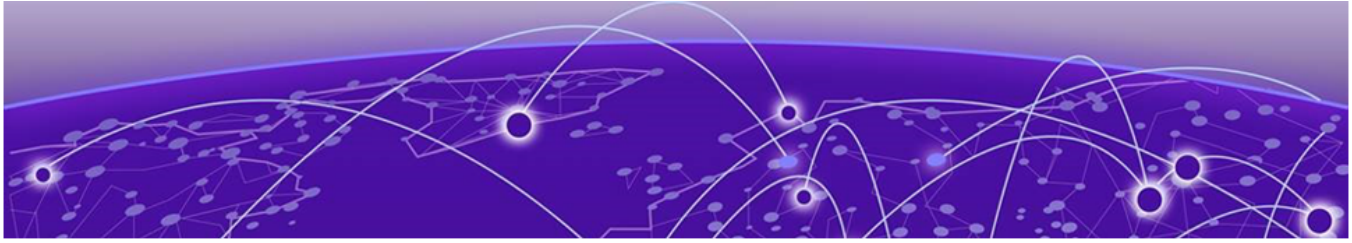
The RESTCONF API returns standard HTTP status codes in addition to JSON-based error codes and messages in the response body.

Table 11: 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

Table 11: HTTP Status Codes (continued)

Code	Description
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



Log in to the RESTCONF Server

RESTCONF Root Resource on page 38
Authentication on page 39
Access the RESTCONF Datastores on page 39

You must have admin access to the network device that is running the RESTCONF interface. No separate login credentials are required.

IP address and Ports

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 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

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

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

   Sample client request

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

2. The RESTCONF server responds as follows:

   Sample server response

```
{
    "duration": 86400,
    "token": "eyJhbGciOiJIUzI1NiIsImV4cCI6MTU3MDE5MDk3OSwiaWF0IjoxNTcwMTA0NTc5fQ.
        eyJ1c2VybmFtZSI6ImFkbWluIiwiYWNjZXNzX2xldmVsIjoiYWRtaW4ifQ.
        I2uaCxQ8v5-ShAaZHwUbS76e9LZ22Who4icgLBwmVc8"
}
```

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:vlans/vlan=1/
config \
-H 'Content-Type: application/json' \
-H 'Cookie: x-auth-
token=eyJhbGciOiJIUzI1NiIsImV4cCI6MTU3MDE5MDk3OSwiaWF0IjoxNTcwMTA0NTc5fQ.eyJ1c2VybmFtZS
I6ImFkbWluIiwiYWNjZXNzX2xldmVsIjoiYWRtaW4ifQ.
I2uaCxQ8v5-ShAaZHwUbS76e9LZ22Who4icgLBwmVc8'
```

Access the RESTCONF Datastores

Unlike REST implementations, 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 RETCONF-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 may 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:

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"
            }
          }
        ]
      }
    }
  ]
}
```


Related Topics

ExtremeXOS RESTCONF Reference

API Usage Examples

Get VLAN Details on page 43
Create a VLAN on page 44
Change VLAN Settings on page 44
Delete a VLAN on page 45

This section provides information on how to accomplish some common tasks using the RESTCONF API.

Note
Log in to the RESTCONF server to start a valid session before making any API calls.

Get VLAN Details

To retrieve details of all configured VLANs:

Use the **GET** method to access the `openconfig-vlan` data model.

Sample client request

```
GET /rest/restconf/data/openconfig-vlan:vlans/
    HTTP/1.1
    Host: 10.68.13.192
```

Sample server response

```
{
  "openconfig-vlan:vlans": {
    "vlan": [
      {
        "config": {
          "name": "Default",
          "status": "ACTIVE",
          "tpid": "oc-vlan-types:TPID_0x8100",
          "vlan-id": 1
        },
        "members": {
          "member": [
            {
              "interface-ref": {
                "state": {
                  "interface": "1"
                }
              }
            },
            {
              "interface-ref": {
                "state": {
                  "interface": "2"
                }
              }
```

```
            },
            {
              "interface-ref": {
                "state": {
                  "interface": "3"
                }
              }
            },
            {
              "interface-ref": {
                "state": {
                  "interface": "4"
                }
              }
            },
          ]
        },
        "state": {
          "name": "Default",
          "status": "ACTIVE",
          "tpid": "oc-vlan-types:TPID_0x8100",
          "vlan-id": 1
        },
        "vlan-id": "1"
      }
    ]
  }
}
```

Create a VLAN

To create a VLAN:

Use the **POST** method to add the new VLAN configuration to the `openconfig:vlans/vlan` datasource.

Sample client request

```
POST /rest/restconf/data/openconfig-vlan:vlans/
    HTTP/1.1
    Host: 10.68.13.192
    Body:
       {
        "openconfig-vlan:vlan":[
        {
            "config": {
                "name": "vlan_10",
                "status": "ACTIVE",
                "tpid": "oc-vlan-types:TPID_0x8100",
                "vlan-id": 10
            }
        }
       ]
    }
```

If the request is successful, the server responds with a `201 CREATED` status message.

Change VLAN Settings

To change the VLAN description:

Use the **PATCH** method to change the configuration values for a specific VLAN in the `openconfig-vlan` data model.

Sample client request

```
PATCH /rest/restconf/data/openconfig-vlan:vlans/vlan=10/config
    HTTP/1.1
    Host: 10.68.13.192
    Body:
    { "openconfig-vlan:config": {
    "name": "new_vlan",
    "status": "ACTIVE",
    "tpid": "oc-vlan-types:TPID_0x8100", "vlan-id": 10
    }
}
```

Sample response from server

```
{ "openconfig-vlan:config": {
    "name": "new_vlan",
    "status": "ACTIVE",
    "tpid": "oc-vlan-types:TPID_0x8100",
    "vlan-id": 10
}
}
```

Delete a VLAN

To delete a specific VLAN:

Use the **DELETE** method to delete the specific VLAN.

Sample request from client

```
DELETE /rest/restconf/data/openconfig-vlan:vlans/vlan=10
    HTTP/1.1
    Host: 10.68.13.192
```

If the request is successful, the server responds with a `204 No Content` message.